PROYECTO FIN DE CARRERA

Departamento de Física

Desarrollo de redes neuronales artificiales para el cálculo del transporte neoclásico en reactores de fusión

Autor: Zarza Cano, Estéfano Alfredo
Tutor: Tribaldos Macía, Víctor

Leganés, 1 de junio de 2012

Título: Desarrollo de redes neuronales para el cálculo del transporte neoclásico en reactores de fusión.
Autor: Zarza Cano, Estéfano Alfredo
Tutor: Tribaldos Macía, Víctor

EL TRIBUNAL

Presidente: Raúl Sánchez Fernández (Dpto. de Física)

Vocal: Jesús Iñarrea de las Heras (Dpto. de Física)

Secretaria: Inés Galván León (Dpto. de informática)

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 1 de junio de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIA PRESIDENTE

A MIS PADRES,
ALFREDO Y ELENA.

A MI HERMANO,
ALEXIS.

A MIS AMIGOS,
DAVID, ADRIÁN, ÁLVARO, ÁLEX, BEA, JUANMA Y UN INMENSO ETCÉTERA.

A MI TUTOR,
VÍCTOR.

A MI PROFESOR DE MATEMÁTICAS,
JULIO.

Y A MI NOVIA,
LARA.

La matemática es la ciencia del orden y la medida, de bellas cadenas de razonamientos, todos sencillos y fáciles.

René Descartes

ÍNDICE

1 INTRODUCCIÓN

En multitud de situaciones prácticas es necesario conocer la dependencia de alguna magnitud en función de otras variables y en la mayoría de las ocasiones no se conoce esta dependencia teóricamente y es muy laborioso o caro determinarla experimentalmente. El procedimiento habitual para atacar estas situaciones consiste en estudiar dicha dependencia por medio de *suficientes* datos e intentar ajustar dicha dependencia por medio de alguna función con parámetros libres. El ejemplo más sencillo es el ajuste de rectas por mínimos cuadrados que tanto aplican y odian los estudiantes durante el primer curso en los laboratorios. Desgraciadamente, este método es de muy difícil, sino de imposible, aplicación en situaciones reales debido a que no se conoce la dependencia funcional concreta y por lo tanto los ajustes obtenidos no son precisos y no permiten determinar el comportamiento del sistema en condiciones distintas de las conocidas.

Sin embargo, existe una herramienta matemática, desarrollada en los últimos treinta años, que garantiza, en el sentido matemático del término, la capacidad de ajustar cualquier dependencia funcional con precisión arbitraria: las redes neuronales artificiales (RNA)

En el presente proyecto se presentará esta herramienta matemática, se describirán, brevemente, el procedimiento de ajuste de sus parámetros internos y se aplicará para ajustar una dependencia multifuncional concreta. El ejemplo elegido ha sido el de dos de los coeficientes neoclásicos de transporte que aparecen en los reactores de fusión termonuclear por confinamiento magnético. Tras mostrar el funcionamiento y los resultados obtenidos finalmente se propone una posible aplicación de este mismo procedimiento para controlar el sistema de climatización de una oficina.

El programa con el que vamos a trabajar es MATLAB (MATrix LABoratory). Gracias a sus toolboxes de redes neuronales, este programa se encarga de entrenar las redes con el método que escojamos.

A continuación ofrecemos un breve resumen del contenido de cada capítulo.

1.1 Capítulo 2. Redes Neuronales

En primer lugar mostramos con brevedad la herramienta con la que vamos a trabajar. Partimos primero del modelo biológico y mostramos a continuación las similitudes del modelo artificial con este. En este mismo capítulo también se hace una pequeña introducción a las matemáticas del proceso mostrando su estructura básica e indicando las operaciones que se realizan en cada parte de la red (sumatorios, funciones de transferencia, etc).

Tras presentar los tipos de aprendizaje y las principales funciones de transferencia, mostramos el método de entrenamiento que hemos utilizado, la Backpropagation. En esta parte del capítulo se explican las bases del funcionamiento de este algoritmo, así como sus ventajas e inconvenientes.

Para terminar el segundo capítulo hablaremos de la importancia de la normalización de los datos para facilitar y optimizar el entrenamiento de la red y mostraremos los principales métodos que se utilizan.

1.2 Capítulo 3. Matemáticas del proceso

Este capítulo comienza con una pequeña introducción en la que mostramos las ecuaciones básicas de funcionamiento de nuestra red. La intención es mostrar a lo largo de este cómo se establece la dependencia entre las variables de entrada (estímulos) y la función (salida) por medio de lo que se conoce como *pesos*.

Después explicaremos el método de la Backpropagation y las matemáticas que hay detrás de este algoritmo. Mostramos las reglas que sigue para definir el error de la red y reducirlo al mínimo. En nuestro caso, el método elegido para optimizar la red es el de Levenberg-Marquardt, el cual explicamos también en este capítulo.

1.3 Capítulo 4. Energía de fusión nuclear

A modo de introducción para el problema que trabajamos en este proyecto, hablaremos aquí sobre este tipo de energía.

A partir de la revolución industrial, se han liberado a la atmósfera inmensas cantidades de CO_2 que árboles y plantas habían almacenado durante millones de años, convirtiéndose lentamente en los actuales combustibles fósiles. Debido a esto nos encontramos con dos graves problemas, la imperiosa necesidad de producir energía para abastecer nuestras *necesidades* y la imposibilidad de seguir utilizando combustibles fósiles debido a que se acabarán en un futuro no muy lejano y a la gran contaminación que conlleva su uso desmedido.

Partiendo de esto, mostramos distintas opciones energéticas como son las llamadas energías renovables y la energía nuclear de fisión.

A continuación mostramos los principios de obtención de energía nuclear de fusión y las ventajas y desventajas de ella.

1.4 Capítulo 5. Ajustes

En este capítulo abordaremos el problema del que hablamos al principio de esta presentación.

En un primer apartado nos ponemos en situación comentando el principal inconveniente para obtener energía mediante la fusión nuclear, el confinamiento y transporte de partículas subatómicas a elevadas temperaturas (cientos de millones de grados centígrados). Mostramos los principios que siguen los métodos teóricos (DKES y MOCA) utilizados para obtener los datos que nos han servido como base de datos para nuestro problema, así como el principal inconveniente de estos, el elevado tiempo de trabajo que se necesita para obtener resultados.

Tras esta introducción pasamos a definir nuestro problema mostrando lo que serán nuestras distintas entradas y salidas, así como las normalizaciones sufridas por cada subconjunto de datos. Unas breves explicaciones de ciertas partes del programa que hemos desarrollado nos ayudarán a entender cómo hemos obtenido nuestras redes.

En este capítulo se muestran también los criterios que hemos seguido para elegir una RNA frente a otra.

1.5 Capítulo 6. Resultados obtenidos

Aquí mostramos mediante varias gráficas los resultados enfrentando los datos obtenidos por nuestras redes frente a los teóricos, así como el error calculado como la diferencia entre estos dos, frente a su índice en la base de datos. Se muestran también los errores máximo, mínimo, medio y mediana de cada red.

Por último, hay un pequeño apartado de conclusiones al final de la presentación de los resultados de cada red.

1.6 Capítulo 7. Posibles aplicaciones de las redes neuronales

En este último capítulo proponemos una posible aplicación de las redes neuronales para solucionar otro tipo de problemas.

El problema elegido es el de la climatización de una planta entera de un edificio dedicada al trabajo en oficina. Este tipo de oficina compartida por varias personas es muy común y presentan muchos problemas de climatización ya que dependiendo de la época del año, la orientación del edificio y el lugar dentro de la oficina, la temperatura puede variar mucho de un punto a otro.

La solución que proponemos es un sistema de RNA encargado de controlar la temperatura basándose en las medidas tomadas por varios sensores estratégicamente distribuidos por toda la planta. Además de esto, el sistema tendrá en cuenta el grado de apertura de las distintas compuertas por donde sale el aire, así como la presión en la bomba y la temperatura del fluido a la salida de esta. Por lo tanto, este sistema dista mucho de los convencionales ya que al tener en cuenta todas estas variables, podremos conseguir una temperatura homogénea en toda la planta, independientemente de que el día sea extremadamente frío o soleado.

Para una mejor comprensión del sistema que presentamos se aplica este a un caso ficticio. En un esquema de la planta se muestra la ubicación de sensores, compuertas, ventanas de la sala y ubicación del personal. Definimos tras esto las entradas y salidas

del sistema y mostramos las distintas redes que lo compondrían. Para finalizar explicamos el método de entrenamiento a seguir.

1.7 Apéndices

En los apéndices de este proyecto se muestran los diversos códigos que hemos desarrollado para trabajar con MATLAB y crear las RNA encargadas de obtener los coeficientes anteriormente citados.

En total hay dos apéndices, uno para cada red. Cada uno muestra primero el programa completo con las subrutinas empleadas intercalas, apareciendo a continuación de que el programa las llame. Después se muestran los programas y subrutinas utilizados para evaluar las redes obtenidas y elegir la que obtiene los resultados que mejor se adecuan a nuestras exigencias.

2 REDES NEURONALES

2.1 Introducción

Las redes neuronales son una invención matemática inspirada en observaciones hechas en estudios de sistemas biológicos. Una red neuronal artificial puede ser descrita como el mapeo de un espacio de entrada a un espacio de salida. Este concepto es análogo al de función matemática. El propósito de una red neuronal consiste en asignar una entrada a una salida deseada.

Podemos definir también una red neuronal como una herramienta diseñada para emular la forma en que el cerebro humano funciona.

El cerebro humano está compuesto por una gran cantidad de elementos básicos denominados neuronas. Estas a su vez están formadas básicamente por un cuerpo central (**núcleo**) y un mecanismo de conexión con otras neuronas (**axón** y **dendritas**).

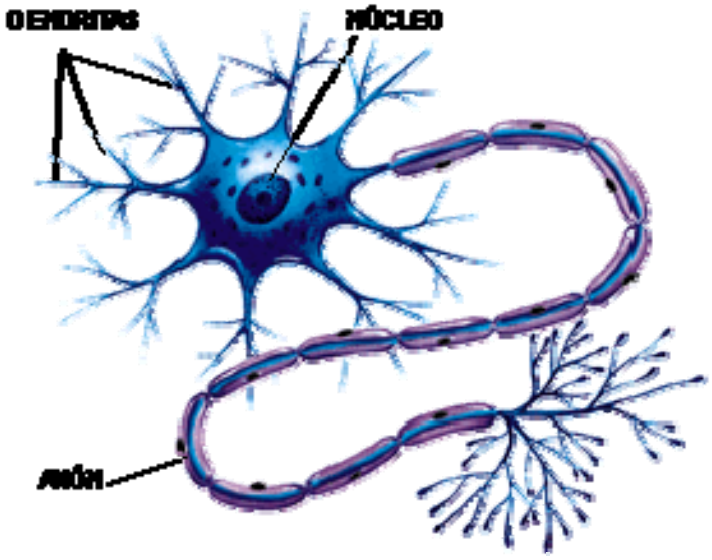

Ilustración 1

Los estímulos recibidos en el cerebro son transmitidos entre las neuronas mediante las conexiones sinápticas. Cuando una neurona es estimulada libera una pequeña cantidad de un componente químico (neurotransmisor). Este viaja a través del axón hasta llegar a las dendritas de otras neuronas en las cuales el proceso se repite. Este proceso sirve para incrementar o disminuir la relación entre las neuronas involucradas en él. Así, **ante un determinado estímulo ciertas neuronas se activan y otras se inhiben**

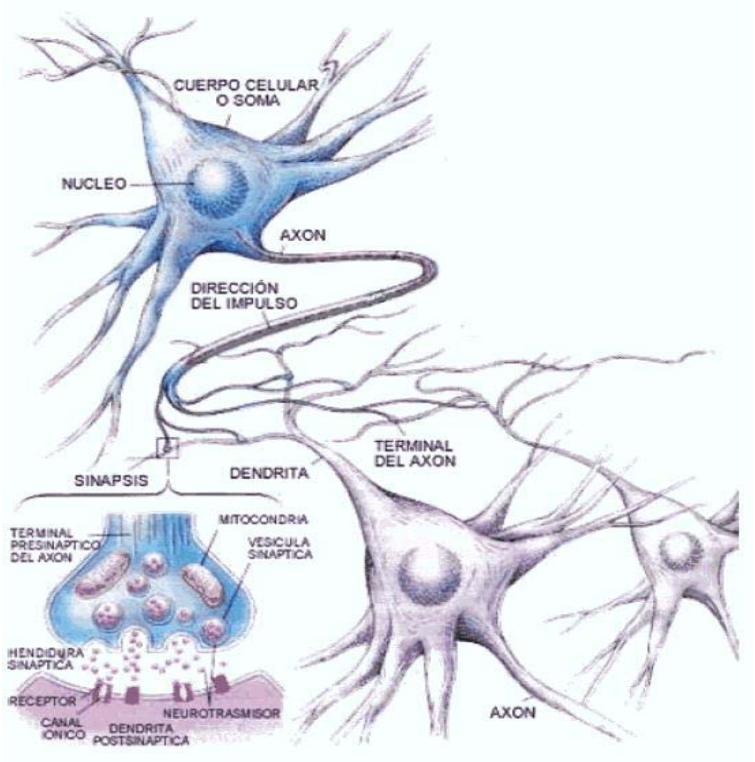

Ilustración 2

Al proceso de creación de nuevas conexiones entre neuronas es a lo que llamamos **aprendizaje**. Cuando este proceso se completa, tenemos **conexiones permanentes** que son en esencia lo que conocemos como **memoria**.

El conocimiento adquirido está entonces en los niveles de relación entre las neuronas logrados durante el proceso de aprendizaje.

Podemos decir pues que el cerebro es entrenado por repetición de estímulos. Mediante un proceso de aprendizaje se logran establecer los niveles correctos de activación-inhibición de las neuronas.

Estudios sobre la anatomía del cerebro humano concluyen que este contiene unas 10^{10} neuronas, cada una con unas $10^{4\text{-}5}$ conexiones sinápticas a la entrada y a la salida de ella. Es este enorme número de neuronas y conexiones el que hace un sistema inigualable a día de hoy en cuanto a aprendizaje e inteligencia como tal se refiere.

2.2 Neuronas artificiales

A mediados del siglo XX nacieron los primeros modelos de neuronas artificiales. Esto impulsó el nacimiento de la cibernética y más tarde nacerá el término Inteligencia Artificial. Estos modelos establecen una relación entre un conjunto de entradas x_i ponderadas por un coeficiente llamado peso w_{ij} y una única salida y_j. Dentro de la neurona hay una función sumatorio y una función de transferencia $f(z)_j$.

Podemos ver en la siguiente ilustración las similitudes entre este modelo de neurona y el modelo biológico.

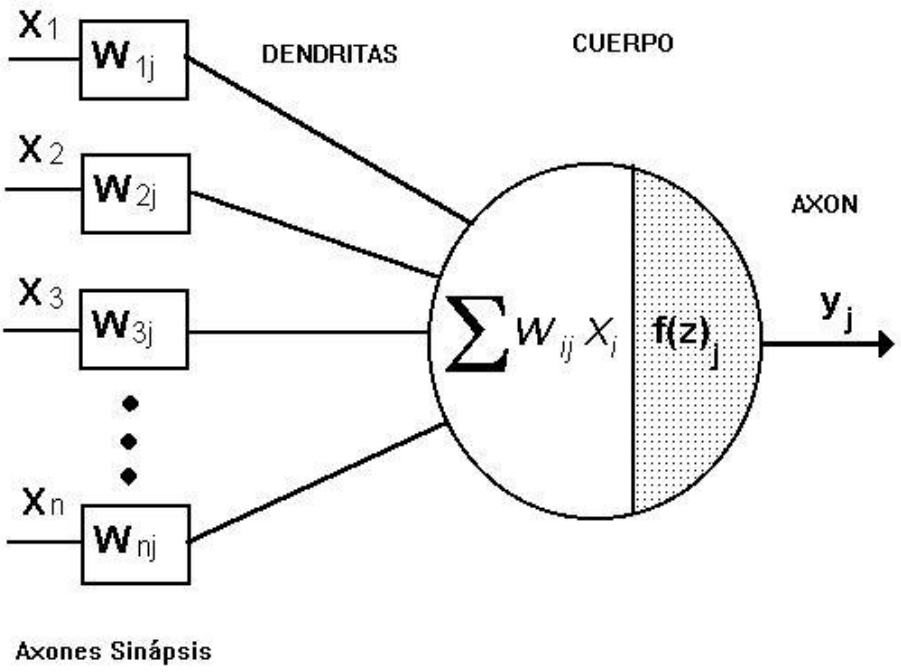

Ilustración 3

Una de las primeras arquitecturas neuronales donde se aplica esta definición de neurona es el "**Perceptrón"**. Este modelo de red neuronal utiliza la siguiente función de salida:

Yj=valor mínimo, si Xj < H
Yj=valor máximo, si Xj>=H

H es una constante denomina umbral. Esta es una función de salida de tipo binaria, llamada en inglés hard limit debido a que es una función tajante que sólo acepta dos valores (-1 y 1 por ejemplo), nada intermedio. La ilustración 4 nos ofrece una gráfica de una función de este tipo. En ella vemos que:

Valor umbral H=0; valor mínimo= -1; valor máximo= +1

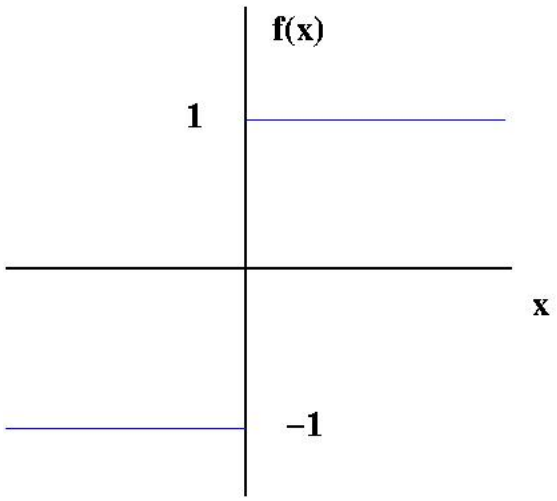

Ilustración 4

Tanto en el Perceptrón como en todos los tipos de redes neuronales, el valor de la salida depende de la entrada y de los pesos que dan mayor o menor relevancia a estas. Los diferentes pesos de una neurona se obtienen mediante un proceso de entrenamiento

de la red. Esta es la gran diferencia entre una red neuronal y una máquina algorítmica clásica, una red no se programa, se "educa".

Una vez que un conjunto de neuronas está interconectado, ¿cómo hacer que la red haga lo que nosotros queremos? ¿Qué peso debe ser modificado y por cuánto? Estas preguntas están relacionadas con el llamado "problema de asignación de crédito", que se puede definir como el proceso de determinar qué peso debe ser ajustado para efectuar el cambio necesario para obtener el rendimiento deseado del sistema. Si las cosas van bien, al peso seleccionado se le da "una palmadita en la espalda", se refuerza la respuesta deseada. Del mismo modo, cuando las cosas van mal, el peso se identifica como culpable y es penalizado. Otras preguntas que necesitan ser respondidas se refieren al número de neuronas a poner en cada capa, el número de capas ocultas, y la mejor manera de entrenar a la red.

Hay **tres tipos de aprendizaje** de redes neuronales:

- *Aprendizaje supervisado:* consiste en introducir una serie de patrones de entrada a la red y a su vez mostrar la salida que se quiere tener. La red es capaz de ajustar los pesos de las neuronas de forma que a la presentación posterior de esos patrones de entrada la red responde con la salida memorizada.
- *Aprendizaje no supervisado:* se presentan los patrones de entrada a la red y ésta los clasifica en categorías según sus rasgos más sobresalientes.
- *Aprendizaje autosupervisado:* la propia red corrige los errores en la interpretación empleando una realimentación.

El Perceptrón, utiliza un aprendizaje supervisado. Trabaja con patrones de entrada binarios, y su funcionamiento, por tratarse de una red supervisada, se realiza en dos fases: En la primera se presentan las entradas y sus correspondientes salidas deseadas. En esta fase la red aprende qué salida tiene que ofrecer cuando se le introduce una entrada u otra. En la segunda fase se comprueba cómo de bien ha aprendido la red ofreciéndole patrones de entrada con una salida conocida con los cuales no ha sido entrenada la red.

La principal aportación del Perceptrón es que la adaptación de los pesos se realiza teniendo en cuenta el error entre la salida que da la red y la salida que se desea.

Otros dos modelos de entrenamiento supervisado basados en el Perceptrón son los llamados “Adaline” y “Madaline”.

Estos modelos usan una regla de aprendizaje llamada “Delta” y funciona de la siguiente manera. Se define una función error para cada neurona como la diferencia entre el valor de salida obtenido por esta al introducirle un conjunto de entradas y la salida deseada ya conocida. Así, la regla de aprendizaje hace que la variación de los pesos se produzca en la dirección y sentido contrario al vector gradiente del error.

Por otra parte, hay muchas clases de problemas poco adecuados para los clasificadores lineales, por lo que la comunidad de la red neuronal se ha visto obligada a trabajar para encontrar una mejor solución para los problemas de tipo linealmente no separables. Esta demostró que mediante la adición de capas de neuronas y con mucho cuidado en la elección de sus funciones de transferencia, el problema de asignación de crédito podría ser resuelto y superar los problemas señalados anteriormente. De este modo, mediante el uso de funciones de transferencia monótonas y continuas, se resuelve el problema de asignación de crédito. Esto se conoce a menudo como gradiente descendente.

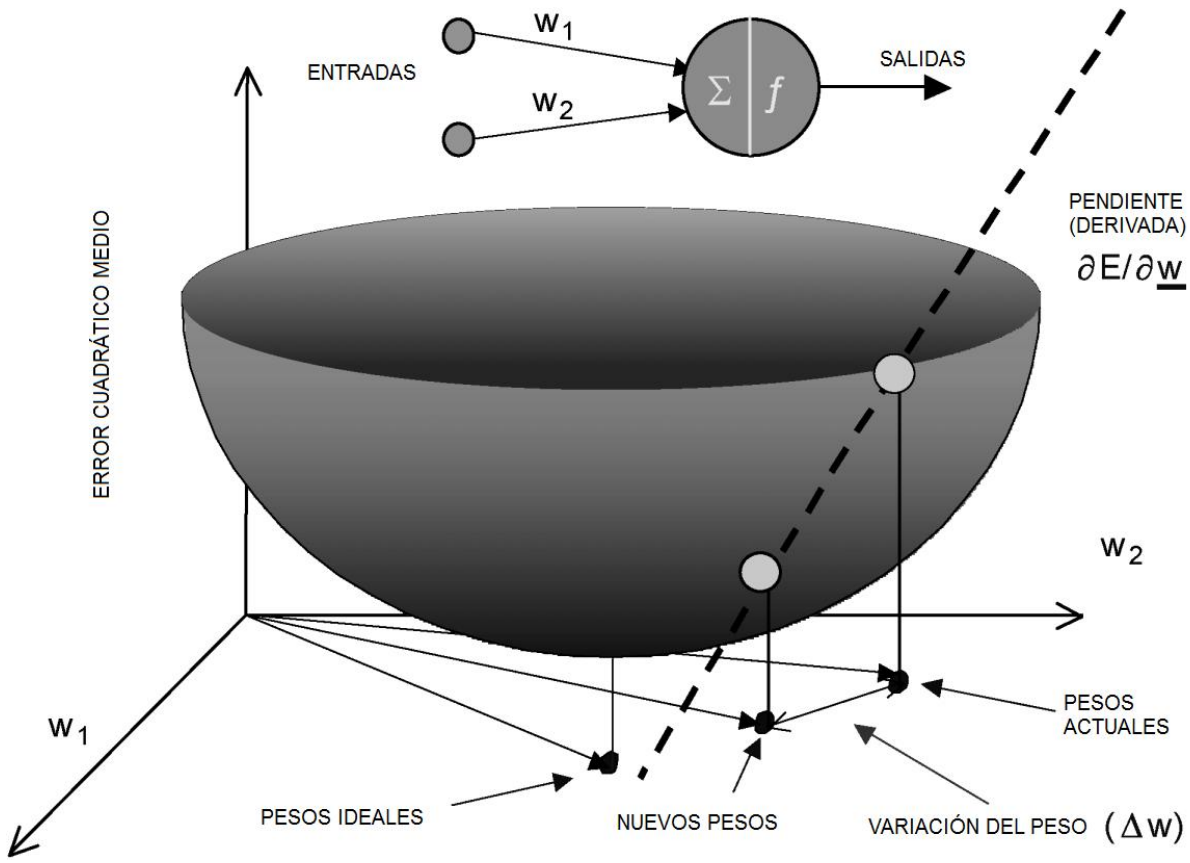

Ilustración 5

Estos resultados permitieron a los investigadores añadir cualquier número de capas ocultas entre la entrada y salida. Aunque se descubrieron nuevos problemas, como que la validez de la solución era directamente proporcional al tiempo de entrenamiento de la red, así como una propensión a encontrar mínimos locales en el espacio de la solución general, lo que hace que la red no converja a una solución óptima.

De la mano de estas aportaciones nació la Backpropagation o propagación hacia atrás. Esta es una técnica de aprendizaje que compara la salida que ofrece la red con la deseada. La diferencia entre ambas constituye un error que se propaga hacia atrás desde la capa de salida hasta la de entrada permitiendo así la adaptación de los pesos de las neuronas intermedias mediante la regla de aprendizaje Delta. Más adelante detallaremos este método más a fondo.

La categoría de redes no supervisadas, se diferencia con otras redes, en que las neuronas que representan patrones parecidos aparecen juntas en el espacio salida, este

espacio puede ser unidimensional, una línea, bidimensional, un plano o N-dimensional. Es el propio diseñador de la red el que establece el espacio de salida que tendrá la red.

2.3 Funciones de transferencia

Como hemos dicho en el apartado anterior, las neuronas envían señales a otras neuronas a través de una señal eléctrica a lo largo del axón. Esto se modela a través del uso de una función de transferencia que imita la "tasa de disparo" de una neurona, como se muestra en la siguiente figura.

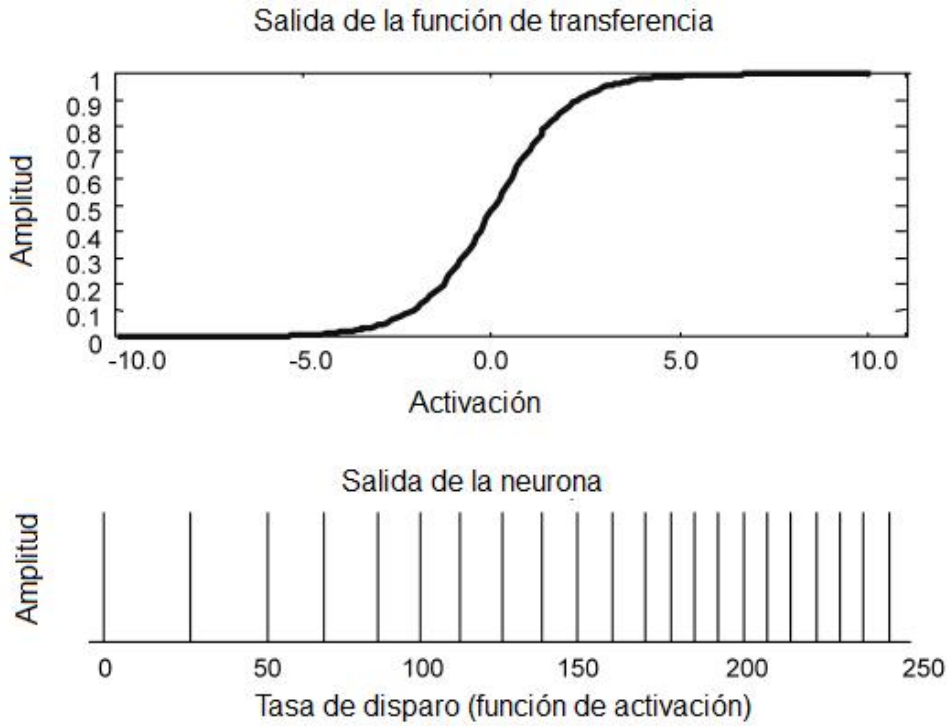

Ilustración 6

Algunas entradas a la neurona pueden tener más relevancia que otras, por lo que debería dársele una mayor importancia. Esto se modela mediante la ponderación las entradas a la neurona con los pesos. Por lo tanto, la neurona puede imaginarse como una pequeña máquina a la que llegan las entradas, esta las procesa, y luego transmite una salida en función de dichas entradas. La neurona artificial con entradas ponderadas y una función de transferencia correspondiente se muestra en la siguiente ilustración.

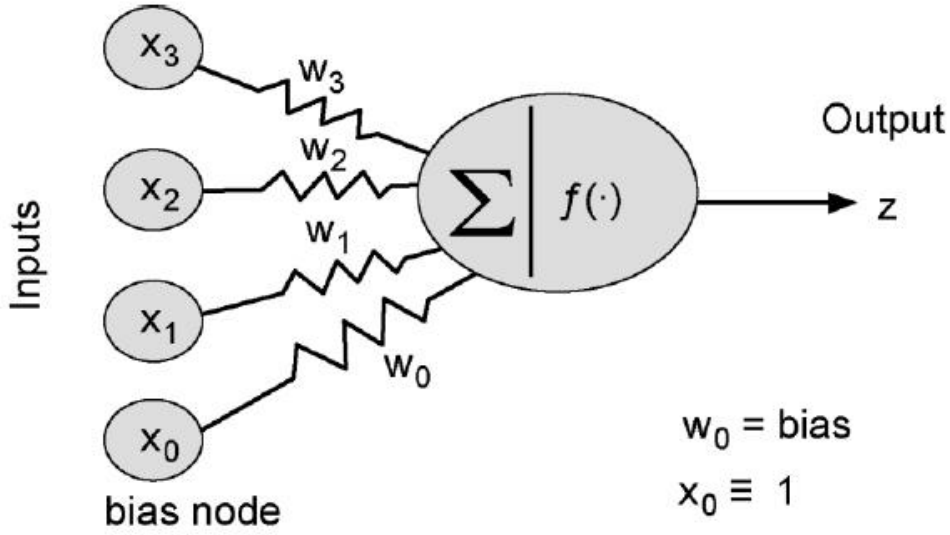

Ilustración 7

En esta ilustración aparece un nuevo término que anteriormente no hemos nombrado, el bias o sesgo. El bias es una cantidad constante que incrementa o disminuye el sumatorio de todas las entradas ponderadas en el nodo de la neurona. A menudo el bias ha sido designado como un peso proveniente de una entrada de valor unitario y se denota como w_0 o θ.

La función de transferencia más común es la función sigmoide, que viene dada por la siguiente ecuación:

$$output = \frac{1}{1+e^{-Estímulo}} \qquad \textbf{(Ecuación 1)}$$

El estímulo que recibe la neurona es:

$$Estímulo = \sum_i w_i x_i + \theta \qquad \textbf{(Ecuación 2)}$$

donde i es el índice de las entradas a la neurona, x_i es la entrada a la neurona, w_i es el peso asignado a esa entrada, y w_0 es el sesgo de la neurona.

Por lo tanto la ecuación 1 quedaría de la siguiente manera:

$$output = \frac{1}{1 + e^{-\left(\sum_i w_i x_i + \theta\right)}} \qquad \textbf{(Ecuación 3)}$$

En la siguiente imagen podemos ver los tres tipos más comunes de funciones de transferencia. Primero entre 0 y 1, y después entre -1 y 1.

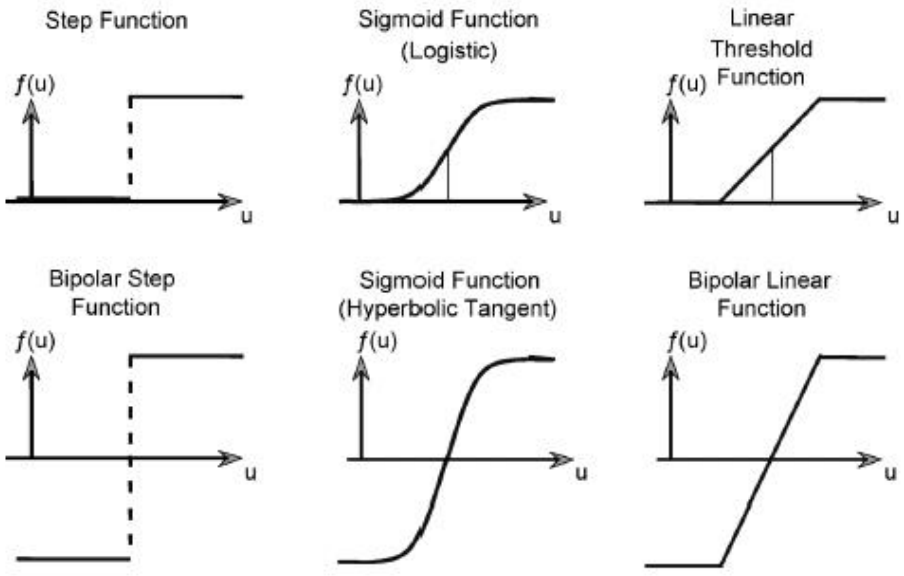

Ilustración 8

2.4 Backpropagation

Durante muchos años no se obtuvo ningún tipo de éxito en el diseño de algoritmos de entrenamiento de redes multicapa. A partir de la comprobación de la severa limitación de los sistemas de una capa, el mundo de la computación neuronal entró en un oscurecimiento y abandono casi general durante dos décadas.

La invención del algoritmo de Backpropagation ha desempeñado un papel vital en el resurgimiento del interés de las redes neuronales artificiales. Este algoritmo es un método de entrenamiento de redes multicapa. Su potencia reside en su capacidad de entrenar capas ocultas y de este modo supera las posibilidades restringidas de las redes de una única capa.

La unidad procesadora básica de la red Feedforward, que usa la Backpropagation como método de entrenamiento y que es la que utilizaremos en MATLAB, se representa en la Ilustración 9. Las entradas se muestran a la izquierda, y a la derecha se encuentran unidades que reciben la salida de la unidad procesadora situada en el centro de la figura.

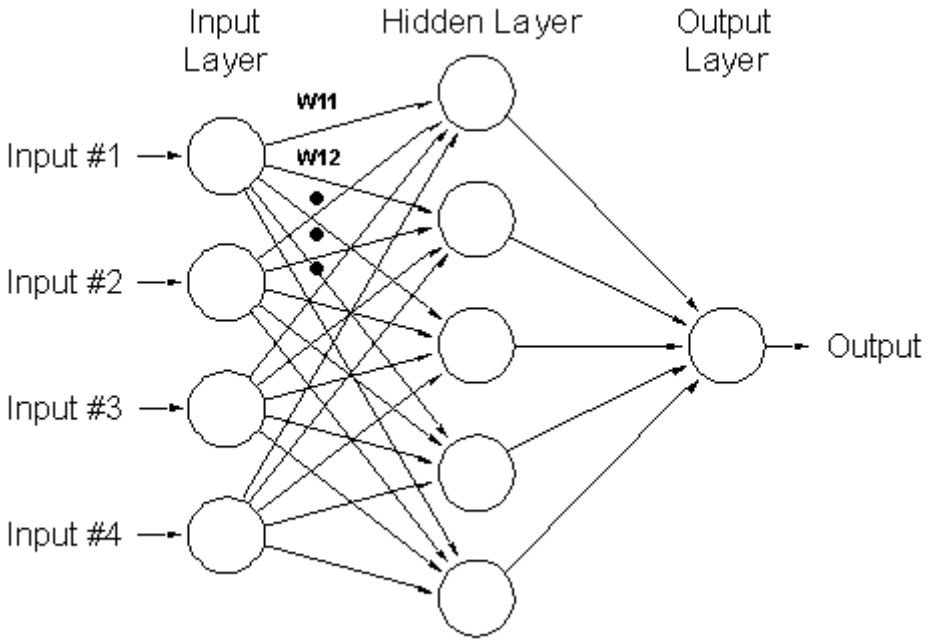

Ilustración 9

La unidad procesadora se caracteriza por realizar una suma ponderada de las entradas, presentar una salida y tener un valor asociado que se utilizará en el proceso de ajuste de los pesos. El peso asociado a la conexión desde la unidad *i* a la unidad *j* se representa por *wij*, y es modificado durante el proceso de aprendizaje.

Normalmente, la Backpropagation utiliza tres o más capas de unidades procesadoras. La ilustración muestra la topología típica de tres capas. La primera capa es la capa de entrada, y se caracteriza por ser la única capa cuyas unidades procesadoras reciben entradas desde el exterior. Sirven como puntos distribuidores, no realizan ninguna operación de cálculo, son las unidades procesadoras de las demás capas las que procesan las señales. La siguiente capa superior es la capa oculta, y todas sus unidades

procesadoras están interconectadas con la primera y última capa. La última capa es la capa de salida que presenta la respuesta de la red.

Backpropagation es un método de entrenamiento supervisado. A la red se le presentan parejas de patrones, un patrón de entrada emparejado con un patrón de salida deseada. Por cada presentación los pesos son ajustados de forma que disminuya el error entre la salida deseada y la respuesta de la red.

El algoritmo de aprendizaje Backpropagation conlleva una fase de propagación hacia adelante y otra fase de propagación hacia atrás. Ambas fases se llevan a cabo por cada patrón presentado en la sesión de entrenamiento.

- *Propagación hacia Adelante:*

Esta fase de propagación hacia adelante se inicia cuando se presenta un patrón en la capa de entrada de la red. Cada unidad de entrada se corresponde con un elemento del vector patrón de entrada. Las unidades de entrada toman el valor de su correspondiente elemento del patrón de entrada y se calcula el valor de activación o nivel de salida de la primera capa. A continuación las demás capas realizarán la fase de propagación hacia adelante que determina el nivel de activación de las otras capas.

Conviene recordar que las unidades procesadoras de la capa de entrada no realizan ninguna operación de cálculo con sus entradas, ni operaciones con funciones umbrales, sólo asumen su salida como el valor del correspondiente elemento del vector de entrada.

Por otro lado, algunas de estas redes utilizan bias como parte de cualquiera de las capas ocultas y de la capa de salida. Estas unidades presentan constantemente un nivel de activación de valor 1. Además esta unidad está conectada a todas las unidades de la capa inmediatamente superior y los pesos asociados a dichas conexiones son ajustables en el proceso de entrenamiento. La utilización de esta unidad tiene un doble objetivo, mejorar las propiedades de convergencia de la red y ofrecer un nuevo efecto umbral sobre la unidad que opera.

- *Propagación hacia Atrás:*

Una vez se ha completado la fase de propagación hacia adelante se inicia la fase de corrección o fase de propagación hacia atrás. Los cálculos de las modificaciones de

todos los pesos de las conexiones empiezan por la capa de salida y continua hacia atrás a través de todas las capas de la red hasta la capa de entrada. Dentro de los tipos de ajuste de pesos se puede clasificar dos grupos, ajuste de unidades procesadoras de la capa de salida y ajuste de unidades procesadoras de las capas ocultas.

Ajuste de Pesos de la Capa de Salida: el ajuste de estos pesos es relativamente sencillo debido a que existe y se conoce el valor deseado para cada una de las unidades de la capa de salida. Cada unidad de la capa de salida produce un número real como salida y se compara con el valor deseado especificado en el patrón del conjunto de entrenamiento.

A partir del resultado de la comparación se calcula un valor de error para cada unidad de la capa de salida.

Ajuste de Pesos de las Capas Ocultas: estas capas no tienen un vector de salidas deseadas y por tanto el cálculo de este error será más complicado

Convergencia: en el proceso de entrenamiento o aprendizaje de la Backpropagation es frecuente medir cuantitativamente el aprendizaje mediante el error cuadrático medio de la red. Esta medida refleja el modo en el que la red está logrando respuestas correctas; a medida que la red aprende, su error cuadrático medio decrece.

Debido a que los valores de salida de la red y los valores de salida deseados son valores reales, es necesario definir un parámetro de corte o un valor umbral del error cuadrático medio de la red que permita decir que la red se aproxima a la salida deseada y considerar que la respuesta es correcta.

La convergencia es un proceso en el que el error cuadrático medio de la RNA tiende cada vez más al valor 0. La convergencia no siempre es fácil de conseguir porque a veces el proceso puede requerir un tiempo excesivo o bien porque la red alcanza un mínimo local y deja de aprender.

- *Ventajas e inconvenientes:*

La principal ventaja de la Backpropagation es su capacidad genérica de mapeo de patrones. La red es capaz de aprender una gran variedad de estas relaciones. No requiere

un conocimiento matemático de la función que relaciona los patrones de la entrada y los patrones de salida. La Backpropagation sólo necesita ejemplos de mapeo para aprender. La flexibilidad de esta red se ve aumentada con la posibilidad de elegir el número de capas, interconexiones, unidades procesadoras, constante de aprendizaje y representación de datos. Como resultado de estas características este tipo de red es capaz de participar con éxito en una amplia gama de aplicaciones.

El mayor inconveniente es el tiempo de convergencia. Las aplicaciones reales pueden llegar a tener miles de ejemplos en el conjunto de entrenamiento y ello requiere días de tiempo de cálculo. Además la Backpropagation es susceptible de fallar en el entrenamiento, es decir, la red puede que nunca llegue a converger.

Existe una variedad de técnicas desarrolladas para disminuir el tiempo de convergencia y evitar los mínimos locales. Entre ellas destacan cambiar la red, cambiar el conjunto de entrenamiento y añadir ruido aleatorio a los pesos.

2.5 Normalización de los datos

Una de las herramientas más comunes utilizadas por los diseñadores de sistemas de reconocimiento automático para obtener mejores resultados es la normalización de los datos. Hay muchos tipos de normalización de datos. Lo óptimo y por lo tanto lo que se pretende con esto, es que tanto las entradas como las salidas se concentren en un rango de longitud aceptable y estén dispersas dentro de ese rango para un tratamiento de los datos más sencillo.

El método de trabajo con normalización es el siguiente:

1º Normalización de los datos previos de entrada.

2º Normalización de los datos previos de salida.

3º Entrenamiento de la red con los datos normalizados.

4º Utilización de la red para obtener resultados.

Es importante destacar que la red trabaja con entradas normalizadas y devuelve salidas también normalizadas, por lo tanto será necesario agregar dos algoritmos extra a la red en cuestión. Uno de ellos normalizará las entradas que se introduzcan en la red y el otro deberá desnormalizar las salidas que se obtengan.

A continuación vamos a exponer brevemente los tipos de normalización de datos más comunes:

- *Normalización estadística (Z-score Normalization):*

$$x'_i = \left[\frac{(x_i - \mu)}{\sigma}\right] \qquad \textbf{(Ecuación 4)}$$

Donde:

x'_i = Entrada normalizada de índice *i*

x_i = Entrada de índice *i*

μ = Media de las entradas

σ = Desviación típica de las entradas

- *Normalización Min-Max:*

$$x'_i = (\max_{objetivo} - \min_{objetivo}) \times \left[\frac{(x_i - \min_{valor})}{(\max_{valor} - \min_{valor})}\right] + \min_{objetivo} \qquad \textbf{(Ecuación 5)}$$

Donde:

x'_i = Entrada normalizada de índice *i*

x_i = Entrada de índice *i*

$\max_{objetivo}$ = valor máximo del rango que quiero aplicar a las entradas

$\min_{objetivo}$ = valor mínimo del rango que quiero aplicar a las entradas

$\max_{valor}$ = valor máximo real de las entradas

$\min_{valor}$ = valor mínimo real de las entradas

- *Normalización Sigmoidea:*

$$x'_i \equiv \frac{1}{1 + e^{-\left(\frac{(x_i - \mu)}{\sigma}\right)}} \qquad \textbf{(Ecuación 6)}$$

$$x'_i \equiv \frac{1 - e^{-\left(\frac{(x_i - \mu)}{\sigma}\right)}}{1 + e^{-\left(\frac{(x_i - \mu)}{\sigma}\right)}}$$ **(Ecuación 7)**

En la ecuación 6 utilizamos una función logística, la cual coloca los datos normalizados de 0 a 1. En la ecuación 7, por el contrario, utilizamos una función hiperbólica tangencial, la cual coloca los datos de -1 a 1.

Donde:

x'_i = Entrada normalizada de índice *i*

x_i = Entrada de índice *i*

μ = Media de las entradas

σ = Desviación típica de las entradas

3 MATEMÁTICAS DEL PROCESO

3.1 Introducción

La capa de entrada, también conocida como la capa cero, de la red sirve para redistribuir los valores de entrada y no realiza ningún procesamiento. Las salidas de esta capa se describen matemáticamente por la Ecuación 8 donde x_i representa el vector de entrada y N representa el número de neuronas en la capa de entrada:

$$o_i = x_i, \qquad donde \quad i = 1,...,N \qquad \textbf{(Ecuación 8)}$$

La entrada para cada neurona en la primera capa oculta de la red es la suma ponderada de todas las conexiones entre la capa de entrada y la neurona en la capa oculta. Esta suma ponderada es llamada a veces *estímulo* o *entrada de red*. Podemos escribir la entrada de red de una neurona de la primera capa como el producto entre el vector de entrada, x_i, y el peso, w_i, además del bias, θ. La aportación total ponderada, o estímulo neto, a la neurona es la suma de estas señales de entrada individuales y está descrito en la Ecuación 9:

$$Estímulo = \sum_{i=1}^{N} w_i x_i + \theta \qquad \textbf{(Ecuación 9)}$$

El estímulo de red se transforma por la activación de la neurona o de la función de transferencia, *f (Estímulo)*, para producir un nuevo valor de salida de la neurona.

Con Backpropagation, esta función de transferencia es comúnmente una sigmoide o una función lineal. Además de los estímulos de red, el bias, θ, se añade generalmente para compensar la entrada. Por lo tanto, el resultado final de la neurona viene dado por la siguiente ecuación y en la Ilustración 10 se muestra gráficamente el proceso de sumar entradas ponderadas dentro de una neurona y luego aplicar una función de activación para producir una salida.

$$Salida = f\left(Estímulo\right) = f\left(\sum_{i=1}^{N} w_i x_i + \theta\right) \qquad \textbf{(Ecuación 10)}$$

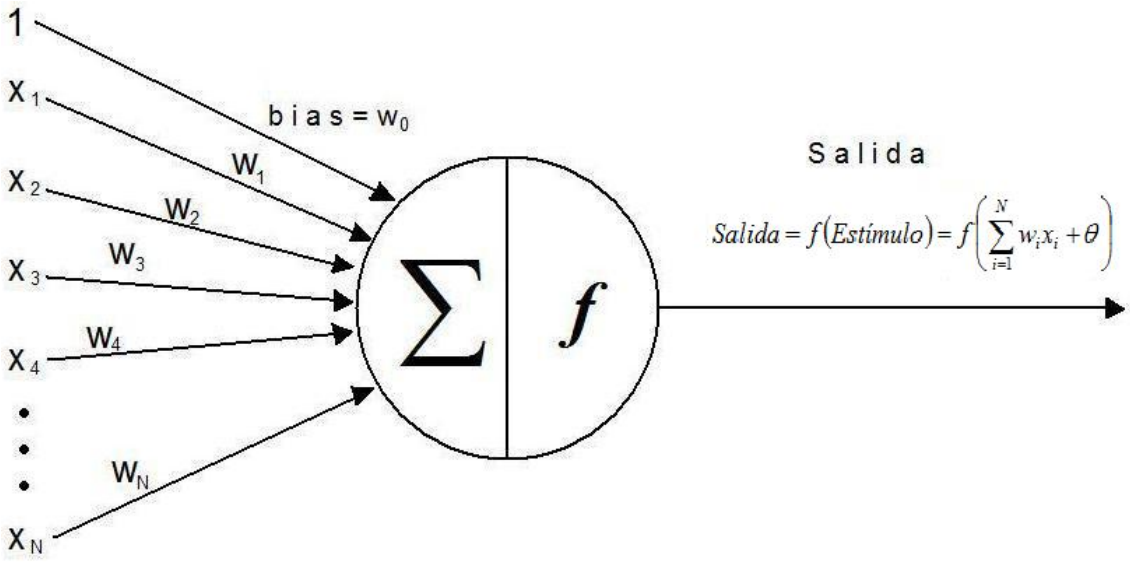

Ilustración 10

Para una red con más de una neurona en una única capa oculta y con una sola salida, la ecuación pasa a ser de la siguiente manera:

$$Salida\left(x_1,...,x_N\right) = \sum_{j=1}^{M} w_j^O \cdot f\left(\sum_{i=1}^{N} w_{ij}^I x_i + \theta_j^I\right) + \theta^O \qquad \textbf{(Ecuación 11)}$$

donde:

El número de neuronas en la capa oculta viene dado por j y va desde 1 hasta M.

El número de entradas viene dado por i y va desde 1 hasta N.

El superíndice I hace referencia a elementos que conectan la capa de entrada con la capa oculta.

El superíndice O hace referencia a elementos que conectan la capa oculta con la capa de salida.

3.2 Algoritmo de la Backpropagation

Anteriormente hemos mencionado que la Backpropagation es la técnica más comúnmente usado para la formación de una red neuronal supervisada. En esta sección, vamos a ver las matemáticas que hay detrás de este algoritmo. El objetivo de la Backpropagation, al igual que el de la mayoría de este tipo de algoritmos, es ajustar iterativamente los pesos de la red para producir el resultado deseado, reduciendo al mínimo el error en la salida. El objetivo del algoritmo es resolver el problema de asignación de crédito. La Backpropagation utiliza la minimización de las derivadas de primer orden para encontrar una solución óptima (método del gradiente descendente o del descenso máximo). Esta utiliza un conjunto de entrenamiento formado por vectores de entrada, x, y vectores de salida, y. El algoritmo de entrenamiento trata de forzar iterativamente los resultados generados por la red representados por el vector z hacia el vector de salida objetivo deseado, y, mediante el ajuste de los pesos de la red a través del uso de la regla delta generalizada.

- *Regla Delta Generalizada:*

Cuando un vector de entrada se presenta a la red, el error de salida puede ser calculado como el error cuadrático medio. El error cuadrático medio se calcula como la suma de los cuadrados de las diferencias entre los valores objetivo y valores de salida:

$$E = \frac{1}{2}\sum_{l=1}^{L}(y_l - z_l)^2 \qquad \textbf{(Ecuación 12)}$$

donde L es la longitud total del vector de la muestra de datos y l el índice de cada dato dentro de L.

Para reducir el error en la red, minimizamos el error con respecto a los pesos de esta intentando llevarlo a cero. Esta minimización del error del gradiente descendente viene definida por:

$$\frac{\partial E}{\partial\left(w_{ij}^{I}, w_{ij}^{O}, \theta_{j}^{I}, \theta^{O}\right)} \equiv 0 \qquad \textbf{(Ecuación 13)}$$

MATLAB busca para que valores de $w_{ij}^{I}, w_{ij}^{O}, \theta_{j}^{I}, \theta^{O}$ el error es mínimo. El programa se encarga de llevar a cabo esta minimización con métodos como el de Levenberg-Marquardt, que más adelante exponemos.

- *Ventajas y desventajas de la Backpropagation, alternativas:*

Ventajas del algoritmo de Backpropagation
1. Es fácil de usar con pocos parámetros para ajustar.
2. Es un algoritmo fácil de implementar.
3. Es aplicable a un amplio rango de problemas
4. Es capaz de formar arbitrariamente complejos mapeos no lineales.

Desventajas del algoritmo de Backpropagation
1. Hay una incapacidad para saber cómo generar con la máxima precisión un procedimiento de mapeo arbitrario.
2. Es difícil saber cuantas neuronas y capas son necesarias.
3. El aprendizaje puede ser lento.
4. Los nuevos aprendizajes sustituyen a los aprendizajes antiguos a menos que los viejos patrones se repitan en el proceso de formación.
5. No tiene la capacidad de detectar cosas nuevas, sólo cosas iguales o parecidas a las que se han usado para su entrenamiento.

Cuando la Backpropagation se estaba haciendo popular, los ordenadores estaban significativamente más limitados que hoy en día en términos de velocidad y memoria. Varias alternativas fueron desarrolladas para mejorar la velocidad de entrenamiento. Algunas técnicas hacen pequeñas modificaciones en el enfoque de la técnica del gradiente descendente, como la técnica del gradiente conjugado o las técnicas de gradientes de segundo orden.

Técnicas de gradiente de segundo orden:

El algoritmo de Backpropagation computa el jacobiano del error con respecto a los pesos para encontrar la mejor dirección al ajustar estos, luego se aplica una actualización fija del peso de tamaño η, en esa dirección para ajustar los pesos. Las técnicas de gradiente de segundo orden usan el Hessiano, $\partial^2 E / \partial w^2$ del error con respecto a los pesos para adaptar el tamaño del paso en la dirección óptima de actualización. Las técnicas de gradiente de segundo orden tratan básicamente de resolver la siguiente ecuación:

$$\frac{\partial^2 E}{\partial w_{jk}^{l^2}} \equiv 0 \qquad \textbf{(Ecuación 14)}$$

Técnicas basadas en el gradiente de segundo orden son las técnicas de *Quick Propagation, Quasi-Newton* y *Levenber-Marquardt.* Detallaremos esta última ya que es la técnica que vamos a usar para implementar nuestra red.

- *Levenberg-Marquardt:*

El algoritmo de Levenberg-Marquardt (LM) es otro algoritmo de optimización no lineal basado en el uso de las derivadas de segundo orden. Como la Backpropagation, computa los cambios de peso, sin embargo este método es más restrictivo. Los requerimientos de memoria para el algoritmo LM aumentan en función del cuadrado del número de pesos, por lo tanto se limita a redes relativamente pequeñas, del orden de unos cuantos cientos de pesos como mucho.

El algoritmo LM es una combinación entre los métodos de Backpropagation y el método de Newton. Este método supone que la función que se modela es lineal y que el

error mínimo se puede encontrar en un solo paso. Se calcula la variación del peso para hacerlo en un único paso. Testea la red con estos nuevos pesos para determinar donde es más pequeño el error. Una variación en el error sólo es aceptada si esta mejora el error. Cuando el error decrece, la variación del peso es aceptada y la hipótesis lineal se ve reforzada por la disminución de un parámetro de control, μ. Cuando el error aumenta, la variación del peso es rechazada y, como en Backpropagation, se sigue el gradiente para aumentar el control y así restar importancia a la suposición lineal. Cerca de un mínimo, la suposición lineal puede considerarse cierta, por lo que el algoritmo LM trabaja muy rápido. Lejos de un mínimo, la hipótesis lineal suele ser mala, pero al utilizar el gradiente cuando el error no mejora, se conseguirá converger a un error mínimo. El proceso se repite hasta que se consigue el error requerido o se alcanza el número de iteraciones marcado.

El algoritmo de LM se aproxima a la matriz hessiana utilizada en el método quasi-Newton como el producto de una matriz jacobiana de derivadas parciales de primer orden. Ya que utiliza una matriz jacobiana, J, en lugar de una matriz de hessiana, H, el cálculo es más fácil:

$$H \approx J^T J \qquad \textbf{(Ecuación 15)}$$

El gradiente es calculado como el producto del jacobiano y un vector, e, que contiene los errores que han sido minimizados:

$$g = J^T e \qquad \textbf{(Ecuación 16)}$$

Esto nos da la fórmula de actualización del peso en la ecuación 17, donde I es la matriz identidad y μ es el parámetro de control.

$$\Delta w = -\left(J^T J + \mu I\right)^{-1} J^T e \qquad \textbf{(Ecuación 17)}$$

Esta ecuación, cuando μ es 0, se convierte en la ecuación del método de Newton con la aproximación a la matriz hessiana. Para valores muy grandes de μ esta ecuación se aproxima a la ecuación del método del gradiente.

4 ENERGÍA DE FUSIÓN NUCLEAR

Durante millones de años los bosques y selvas del planeta han consumido el CO_2 de la atmósfera interaccionando con el Sol en el proceso de la fotosíntesis. Al morir las plantas han ido dejando en la tierra ese CO_2 que habían cogido de la atmósfera y grandes cantidades de estos restos de plantas han sido enterrados por múltiples capas de tierra y sometidos a elevadas presiones durante millones de años.

En menos de 300 años el ser humano se ha encargado de poner de nuevo en la atmósfera ese CO_2 con graves y obvias consecuencias para el medio ambiente.

A mediados del siglo XVIII comienza la revolución industrial y con ella el auge de los combustibles fósiles. El primero en ser explotado fue el carbón, que a día de hoy sigue siendo la principal fuente de energía en el mundo. Después de este pasamos al petróleo, del que se pueden extraer una infinidad de productos que sirven como materias primas para combustibles (gasolina, gasoil, etc) y productos plásticos e incluso alimenticios.

Hoy en día sería impensable vivir sin los citados combustibles fósiles. Estos han empujado al ser humano a vivir en una nueva era gracias a la gran capacidad energética que poseen, pero a un alto precio. Los residuos que se envían a la atmósfera tras la quema de estos combustibles (CO_2 entre otros) son extremadamente contaminantes y ya han causado graves daños a nuestro planeta y a sus habitantes (agujeros en la capa de ozono, problemas respiratorios, calentamiento global, efecto invernadero, etc). Además de estos graves problemas existe otro no menos grave, pues estos combustibles son finitos y han tardado millones de años en generarse por lo que su pérdida es

irrecuperable. Debido a esto, el ser humano se encuentra con la necesidad de encontrar combustibles alternativos para obtener energía.

Las llamadas *energías renovables* son aquellas que minimizan la producción de residuos perjudiciales para el medio ambiente y que proceden de fuentes de energía *inagotable*, como pueden ser el sol, el viento o las mareas. Este tipo de energía ha evolucionado mucho desde su aparición hasta nuestros días, pero a menudo resulta poco eficiente para obtener grandes cantidades de energía como se obtienen con los combustibles fósiles.

La energía de fisión nuclear, por el contrario, es capaz de obtener grandes cantidades de energía con pequeñas cantidades de combustible. El principal problema son los residuos que quedan tras la obtención de la energía, estos son altamente radiactivos y han de ser confinados en los llamados cementerios nucleares para que permanezcan allí largos periodos de tiempo. Otro problema mayor si cabe se produce cuando se *descontrola* una reacción. Los reactores de fisión funcionan en régimen *subcrítico*, es decir, el número de neutrones que se generan tras la fisión (ruptura) de un núcleo de uranio no son suficientes para garantizar la fisión de otros núcleos vecinos. Mediante el uso de unas barras de grafito es posible controlar el número de neutrones fruto de una reacción y hacer más o menos probable las demás. Insertando dichas barras se puede *detener* el reactor. Los grandes accidentes en las centrales (Three Mile Island, Chernobyl o el último de Fukushima) siempre se deben a problemas en el sistema de refrigeración del reactor y el corte de la corriente eléctrica lo cual deja a la central sin posibilidad de accionar las barras de control. En estas condiciones la ruptura de los núcleos continúa y el agua que contiene la vasija del reactor se calienta debido a las colisiones de los neutrones con los núcleos de oxígeno e hidrógeno del agua. Llega un momento en el que la presión del vapor de agua es tan alta que revienta el contenedor y libera a la atmósfera su contenido. El problema es que dentro no solo hay combustible sin quemar sino también el ya quemado que suele ser inestable y sigue desintegrándose. Debido a esto hay gran controversia sobre su uso y recientemente se ha declarado el cierre de centrales nucleares de este tipo en países como Alemania y España.

Llegamos ahora a la energía obtenida por fusión nuclear. El principio de obtención de esta es básico y consiste en hacer chocar dos átomos con suficiente energía como para que se fundan en uno mayor. Determinados elementos liberan más energía al fundirse que la que necesitan para provocar la fusión. Esto es debido a que el átomo final tiene menos masa que la suma de los dos átomos que lo forman, esta masa que falta es la que pasa a convertirse en energía. La cantidad de energía generada sigue la famosa ecuación de Einstein:

$$E = mc^2 \qquad \textbf{(Ecuación 18)}$$

En la naturaleza sucede este fenómeno constantemente en las estrellas, las cuales transforman ingentes cantidades de hidrógeno en helio. La energía resultante de este proceso es liberada en forma de luz y calor.

Para conseguir esto artificialmente necesitamos excitar muchísimo los átomos para vencer las fuerzas de repulsión que existen entre ellos y acercarlos tanto que las fuerzas electromagnéticas dejen de ser relevantes en comparación con la fuerza nuclear fuerte, la cual hace que los protones y neutrones se mantengan juntos en el núcleo de los átomos. Esto se consigue elevándolos a altísimas temperaturas (cientos de millones de grados centígrados), por lo que es necesario trabajar con materia en estado de plasma.

Son dos isótopos del hidrógeno los que necesitan el menor aporte de energía para vencer las fuerzas de repulsión culombiana, el deuterio y el tritio.

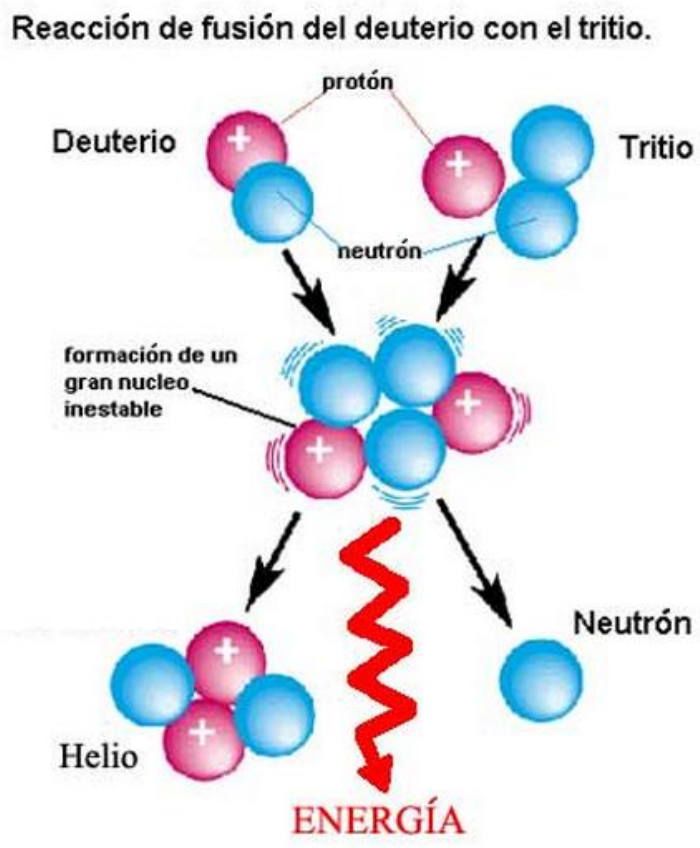

Ilustración 11

Aquí nos encontramos con uno de los principales problemas: no disponemos de ningún recipiente capaz de contener una masa de hidrógeno a estas temperaturas. Actualmente se está experimentado con 2 formas de conseguir el confinamiento del plasma. El confinamiento inercial y el confinamiento magnético, que es el que nos concierne.

Debido a la gran temperatura, los electrones se separan de los núcleos por lo que la materia pasa a ser un gas formado por un conglomerado de estos núcleos (cargados positivamente) y electrones libres (cargados negativamente). Haciendo uso de campos magnéticos se puede limitar la expansión natural de un gas a estas temperaturas gracias a la fuerza de Lorentz.

Las principales ventajas de la fusión nuclear son que los combustibles son prácticamente inagotables (el hidrógeno es uno de los elementos más abundantes en la Tierra), están repartidos uniformemente por la tierra. El deuterio se obtiene del agua, y

el tritio a partir del litio, un elemento bastante abundante. Además, las reacciones son muy estables y controladas, ya que en el momento en el que se deja de aplicar energía para que se lleve a cabo la fusión, el proceso termina. Esto se convierte en uno de los principales inconvenientes, ya que hoy en día es muy complicado realizar una reacción de fusión "rentable", es decir, que genere considerablemente más energía de la que necesita para activarse.

Los otros inconvenientes principales están relacionados con el tritio. A pesar de que el litio es un elemento abundante en la tierra, obtener tritio a partir de este no es sencillo, hoy en día los métodos de obtención de tritio son verdaderos secretos de estado. Por otra parte, este es un isótopo extremadamente volátil y huidizo, como el hidrógeno, y puede sustituir a éste en la formación de moléculas de agua, para dar agua tritiada, muy perjudicial para la salud.

Probablemente en un futuro que todavía no sabemos si es muy lejano o no, la fusión nuclear será una de las fuentes de energía más importantes.

5 AJUSTES

5.1 Introducción

El problema fundamental de la fusión termonuclear controlada es limitar las pérdidas de energía de los diseños actuales de reactores mediante el uso de nuevas estructuras de campo magnético. El estudio del ritmo de pérdida de partículas y energía se denomina transporte.

En el presente proyecto vamos a utilizar las redes neuronales para obtener dos coeficientes que regulan el transporte de partículas subatómicas en un reactor de fusión nuclear mediante el uso de la teoría neoclásica de transporte. Los coeficientes que vamos a hallar sirven para optimizar este proceso y disminuir las pérdidas producidas en el reactor.

Estos coeficientes llamados *coeficientes monoenergéticos de transporte,* han sido hallados con anterioridad por medio de dos códigos numéricos, DKES y MOCA. A pesar de ser unos métodos muy optimizados y partir de muchas hipótesis que simplifican el cálculo de los coeficientes, el tiempo empleado para obtener los coeficientes es muy elevado. Como ejemplo podemos citar que la base de datos empleada en el presente proyecto ha requerido de aproximadamente 50000 horas de cálculo.

Nuestro objetivo será el de ser capaces de obtener todos los valores que queramos de estos coeficientes, que dependen de varias variables de entrada, de una manera casi inmediata y con una fiabilidad muy elevada.

El dato que persigue la teoría neoclásica es el flujo medio de partículas en la dirección radial, la cual se calcula como sigue:

$$\langle \Gamma \cdot \nabla r \rangle = -n\left(L_{11}A_1 + L_{12}A_2 + L_{13}A_3\right) \quad \textbf{(Ecuación 19)}$$

Donde:

n es la densidad del plasma,

A_i viene dado por las siguientes ecuaciones:

$$A_1 = \frac{1}{n}\frac{dn}{dr} - \frac{qE_r}{T} - \frac{3}{2}\frac{1}{T}\frac{dT}{dr} \quad \textbf{(Ecuación 20)}$$

$$A_2 = \frac{1}{T}\frac{dT}{dr} \quad \textbf{(Ecuación 21)}$$

$$A_3 = \frac{qV_L}{R_0 T} \quad \textbf{(Ecuación 22)}$$

Donde

dn/dr es la variación de la densidad en la dirección radial,

qE_r/T es el campo eléctrico radial,

dT/dr es la variación de la temperatura en la dirección radial,

qV_L/R_0T es el campo eléctrico paralelo o potencial eléctrico.

Los L_{ij} vienen definidos por:

$$L_{ij} = \frac{2}{\sqrt{\pi}}\int_0^\infty dK\sqrt{K}e^{-K}D_{ij}h_i h_j \quad \textbf{(Ecuación 23)}$$

Donde $K = mv^2/2T$,

$h_1 = h_3 = 1$,

$h_2 = K$

y D_{ij} son los llamados coeficientes monoenergéticos de difusión.

Los dos que nos competen se calculan de la siguiente manera:

$$D_{11} = -\frac{v_d^2 R_0}{2v}\left\langle \int_{-1}^{1} dp \frac{1}{v_d}\frac{dr}{dt}\hat{f}_{II} \right\rangle \qquad \textbf{(Ecuación 24)}$$

$$D_{33} = -\frac{vR_0}{2}\left\langle \int_{-1}^{1} dp \; p\frac{B}{B_0}\hat{f}_{I} \right\rangle \qquad \textbf{(Ecuación 25)}$$

Donde:

r marca el centro de cada punto,

v es la velocidad de cada punto,

$p = f(v, B)$,

B es el valor del campo magnético para cada punto,

R_0 es el radio mayor,

B_0 es el valor de referencia del campo magnético en la superficie del flujo,

$v_d = mv^2 / (2qR_0B_0)$

q es el valor de la carga,

$\hat{f}_I$ y $\hat{f}_{II}$ son funciones de distribución que dependen de *r, v* y *p*: $f = f(r, v, p)$ y se obtienen de:

$$\frac{R_0}{v}V\left(\hat{f}_I\right) - \frac{R_0}{v}\nu L\left(\hat{f}_I\right) = -p\frac{B}{B_0} \qquad \textbf{(Ecuación 26)}$$

$$\frac{R_0}{v}V\left(\hat{f}_{II}\right) - \frac{R_0}{v}\nu L\left(\hat{f}_{II}\right) = -\frac{1}{v_d}\frac{dr}{dt} \qquad \textbf{(Ecuación 27)}$$

Donde:

ν es la frecuencia de colisionalidad entre partículas (número de veces en que la dirección de una partícula varía 90º debido a las interacciones con otras partículas, por segundo),

V y L son los operadores de Vlasov y Lorentz.

Como podemos observar, los D_{ij} aparecen dentro de integrales, por lo tanto es necesario evaluarlos miles de veces para calcular los L_{ij} y por tanto el flujo medio de partículas en la dirección radial. La red neuronal nos proporcionará un método rápido y fiable para calcular los D_{ij}.

Los D_{ij} que hemos empleado son los siguientes:

- *D_{11}, coeficiente monoenergético de difusión:*

El primer coeficiente a calcular es el llamado *coeficiente monoenergético de difusión,* y se utiliza para calcular la difusión de partículas y energía en la dirección radial debido a la variación radial de la temperatura o la densidad del plasma y el campo eléctrico radial. Se denota como D_{11}, significando el primer 1 que se genera en la dirección radial y el segundo 1 que los causantes son las variaciones radiales.

- *D_{33}, coeficiente monoenergético de conductividad:*

El segundo coeficiente que vamos a calcular es el llamado *coeficiente monoenergético de conductividad*, y se denota como D_{33}. Este coeficiente permite calcular el transporte de partículas y energía en la dirección del campo (axial) debido a la acción del campo eléctrico en la dirección del campo magnético. Como más adelante veremos, para valores de campo eléctrico perpendicular bajo el D_{33} es constante.

Estos coeficientes van a variar en función de unas variables que se convertirán en las entradas de nuestro sistema. Las definimos a continuación:

- Entrada 1, *radio normalizado*: r/a donde a es el radio mayor de la configuración siendo r el radio para el punto concreto que se estudia. El valor del radio normalizado varía entre 0 para el eje magnético y 1 para el borde del plasma. Esta entrada es adimensional.

- Entrada 2, inverso del *recorrido libre medio*: ν/v , esta entrada es el cociente entre la colisionalidad ν y la velocidad a la que se mueve la partícula. Las dimensiones de esta entrada (donde L es longitud y T es tiempo) son (1/T) / (L/T) = 1/L. Describe el inverso de la distancia media que recorren las partículas entre colisiones.

- Entrada 3, *potencial electrostático*: $Er/v\,B$ donde Er es la componente radial del campo eléctrico del plasma, v es la velocidad de la partícula y B el módulo del campo magnético en esa superficie magnética. Como recordamos, la fuerza de Lorentz del magnetismo venía definida por la siguiente ecuación:

$$\vec{F} = q\left(\vec{E} + \vec{v} \times \vec{B}\right) \qquad \textbf{(Ecuación 28)}$$

Donde $\vec{E}$ es el campo eléctrico, $\vec{v}$ la velocidad de la partícula,y $\vec{B}$ es el campo magnético.

Pues bien, el campo magnético hace que las partículas cargadas eléctricamente giren en círculos debido a que este genera una fuerza perpendicular a la velocidad de la partícula y al propio campo. Por tanto, lo que mide la importancia de la fuerza debida al campo eléctrico y al campo magnético es el cociente entre la componente radial del campo Er y el término $|v|\ |B|$. Esta entrada es adimensional.

A continuación se muestran D_{11} y D_{33} obtenidos con métodos teóricos frente a la colisionalidad para varios valores de $Er\,/\,vB$ (cada uno de un color). En la ilustración 13 vemos claramente lo que comentábamos con anterioridad, D_{33} es constante para valores

de campo eléctrico bajo. El asterisco en los coeficientes y la colisionalidad indican que se están usando valores normalizados:

$\nu^* = R_0 \nu / (\iota v)$ donde ι es la transformada del rotacional

$D_{11}^* = D_{11} / D_{11}^P$ donde $D_{11}^P = \pi v_d^2 / 4 v \iota$

$D_{33}^* = D_{33} / D_{33}^{PS}$ donde $D_{33}^{PS} = \nu^2 \langle B^2 \rangle / 3 \nu B_0^2$

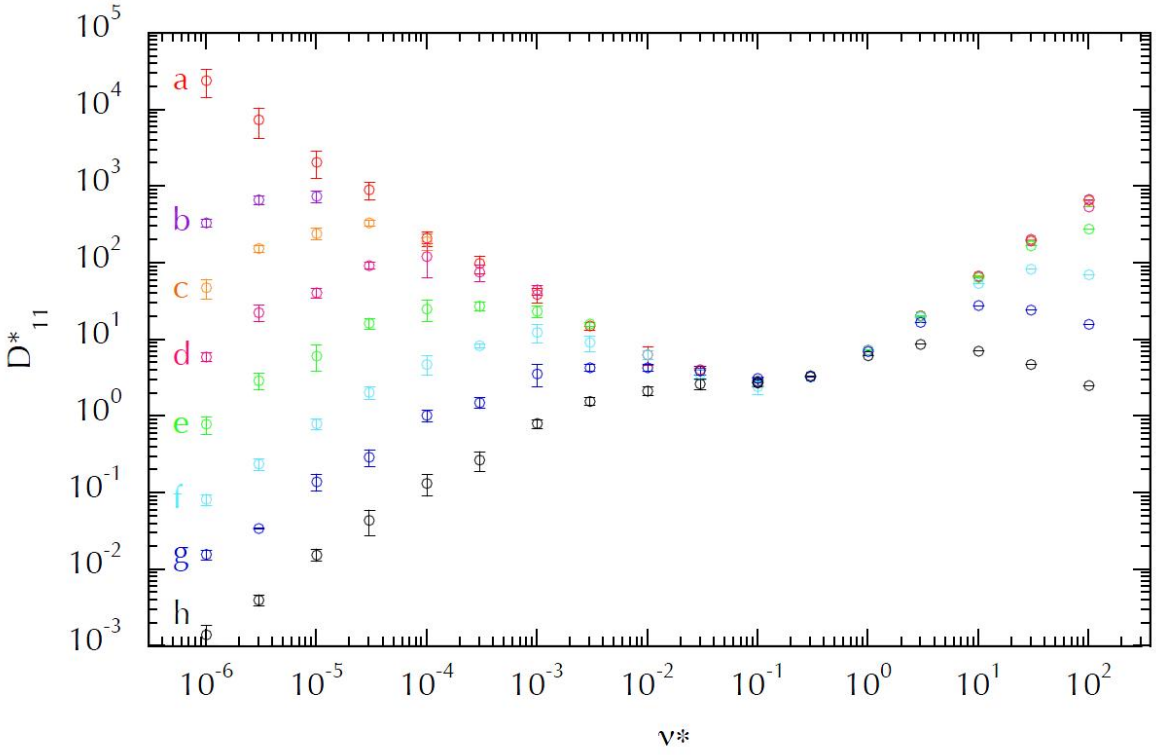

Ilustración 12
Coeficiente monoenergético de difusión frente a la colisionalidad en la mitad del radio de la configuración estándar de TJ-II stellarator, donde: Er/vB = 0 [rojo (a)], 1 x 10-5 [púrpura (b)], 3x10-5 [naranja (c)], 1x10-4[magenta (d)], 3x10-4 [verde (e)], 1x10-3 [azul claro (h)], 3x10-3 [azul(f)], 1x10-2 [negro (g)]. Datos obtenidos con el método MOCA.

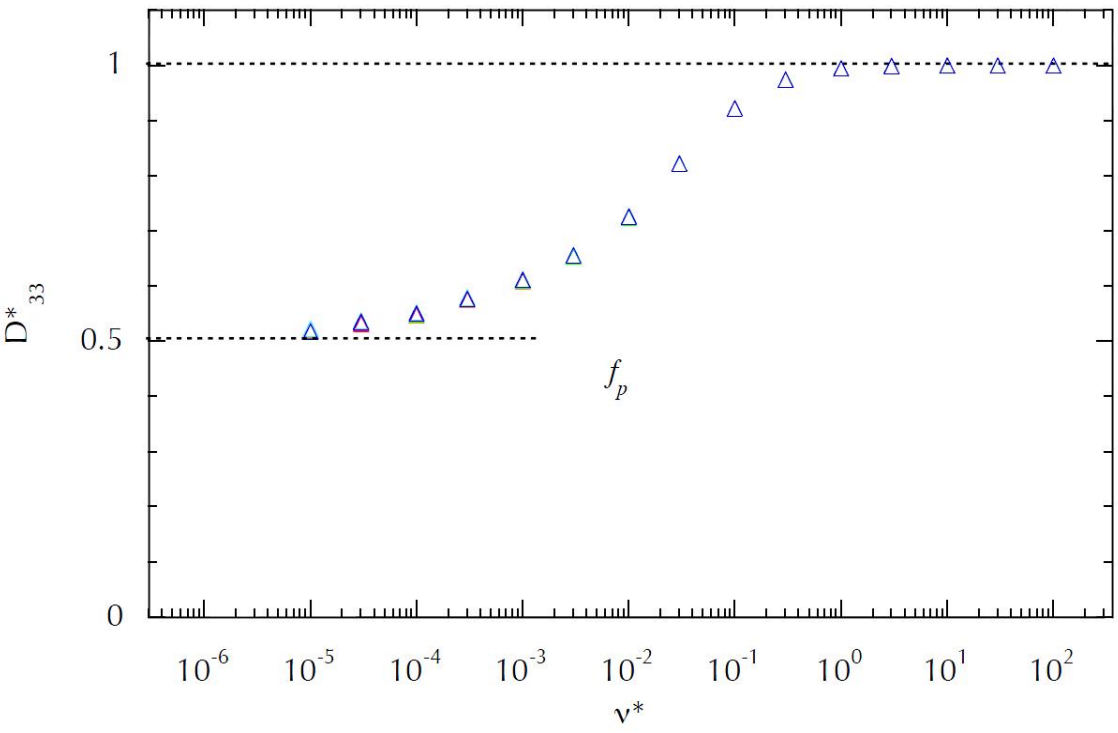

Ilustración 13
Coeficiente monoenergético de conductividad frente a la colisionalidad en la mitad del radio de la configuración estándar de TJ-II stellarator, donde vemos que los valores de D33 no varían con respecto a Er/vB. Datos obtenidos con el método DKES.

5.2 Problema

Como hemos dicho en la introducción, disponemos de una serie de datos y vamos a ajustarlos con dos RNA (una para cada coeficiente). Estos datos se sometieron a un proceso de preparación para facilitar el buen aprendizaje de las redes.El primer paso que hemos dado ha sido el de normalizar los datos, tanto los de entrada como los de salida.

A continuación exponemos unos gráficos que muestran los datos de las distintas entradas y salidas antes y después de normalizarlos. Ha sido necesario tomar logaritmos y ajustar los datos entre unos valores deseados (0 y 1) en el caso del inverso del recorrido libre medio, el potencial electrostático y el coeficiente monoenergético de difusión ya que la dispersión era muy grande y muchos de los datos aparecían *amontonados* en la zona de los valores más pequeños. Al tomar logaritmos y ajustar los datos en un rango relativamente pequeño, conseguimos homogeneizar la repartición de los datos para no perder información y favorecer el rápido entrenamiento de la red.

En el caso del radio normalizado (r/a) y el coeficiente monoenergético de conductividad (D_{33}), no ha sido necesaria ninguna normalización ya que los datos estaban entre 0 y 1 en ambos casos y repartidos de forma homogénea.

Los datos se dibujan frente al índice dentro de la base de datos.

- Entrada 1, *radio normalizado* (r/a):

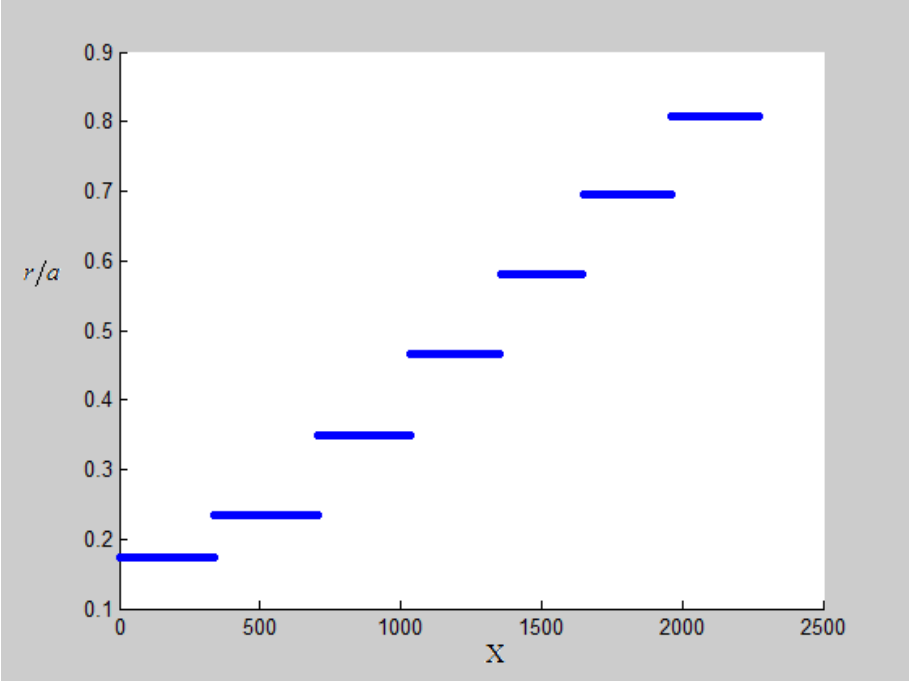

Ilustración 14

Valor mínimo igual a 0,1731. Valor máximo igual a 0.8069.

- Entrada 2, inverso del *recorrido libre medio* (ν/v):

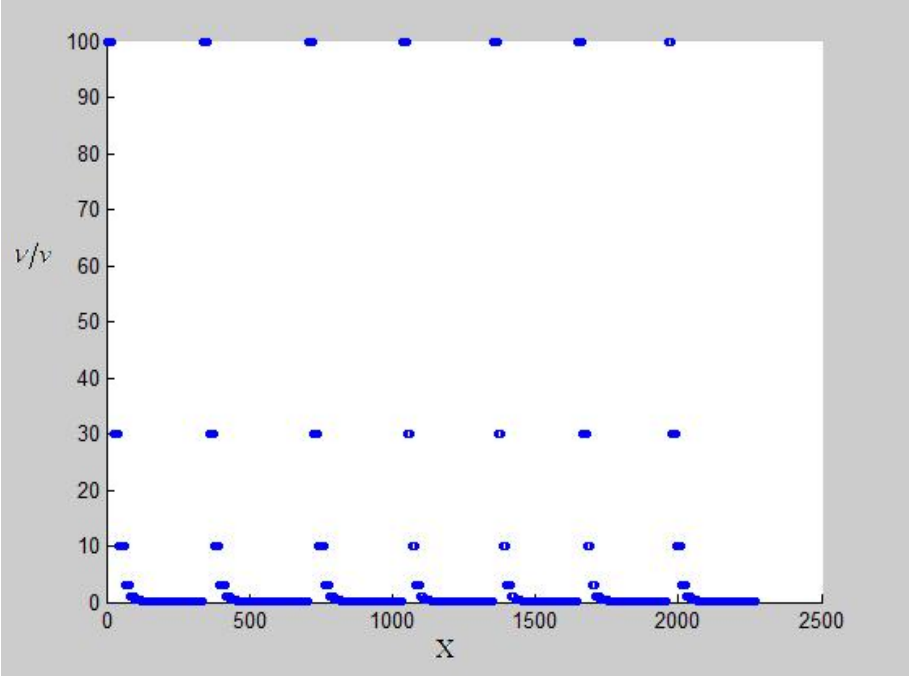

Ilustración 15

Valor mínimo = $3 \cdot 10^{-6}$. Valor máximo = 100.

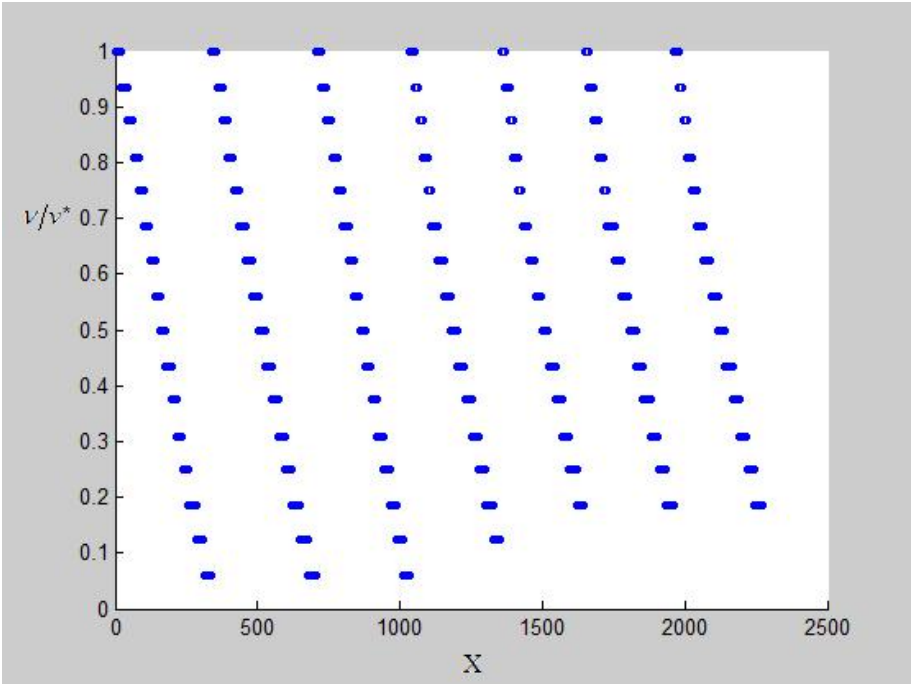

Ilustración 16

Valor mínimo = 0,0596. Valor máximo = 1.

- Entrada 3, *potencial electrostático* (Er/vB):

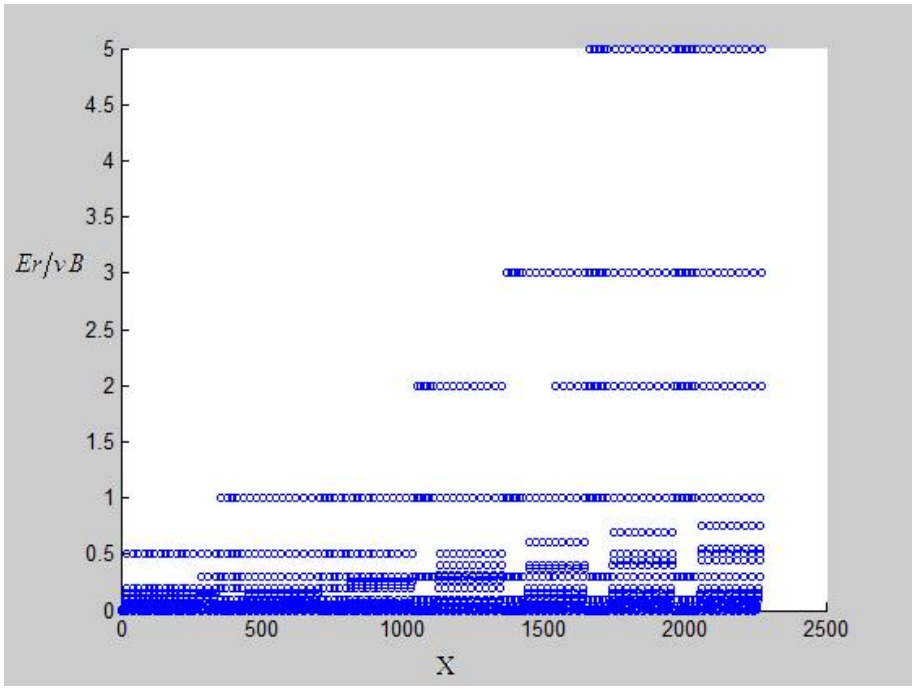

Ilustración 17

Valor mínimo = 0. Valor máximo = 5.

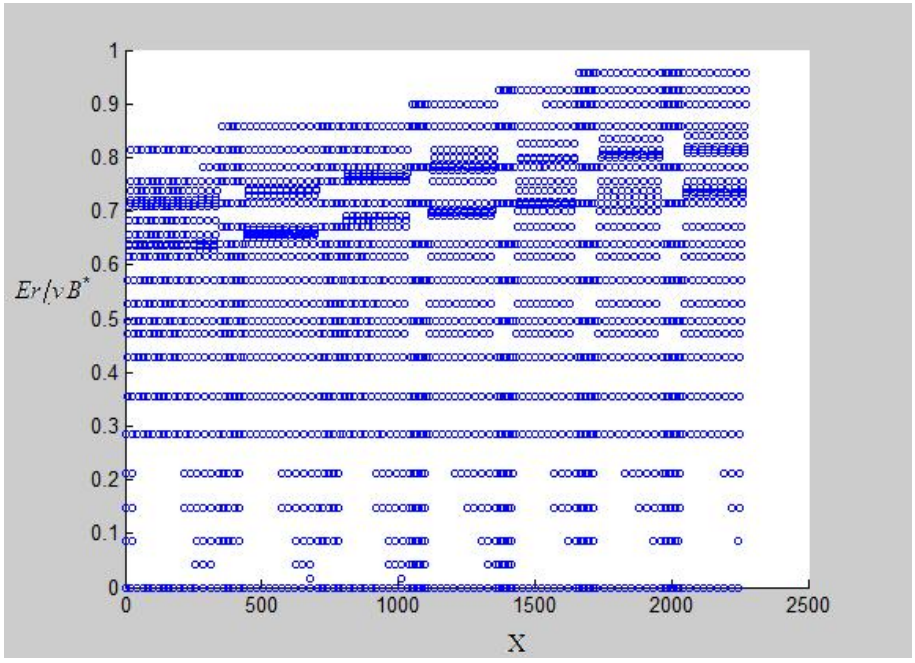

Ilustración 18

Valor mínimo = 0. Valor máximo = 0,957.

- Salida 1, *coeficiente monoenergético de difusión* (D_{11}):

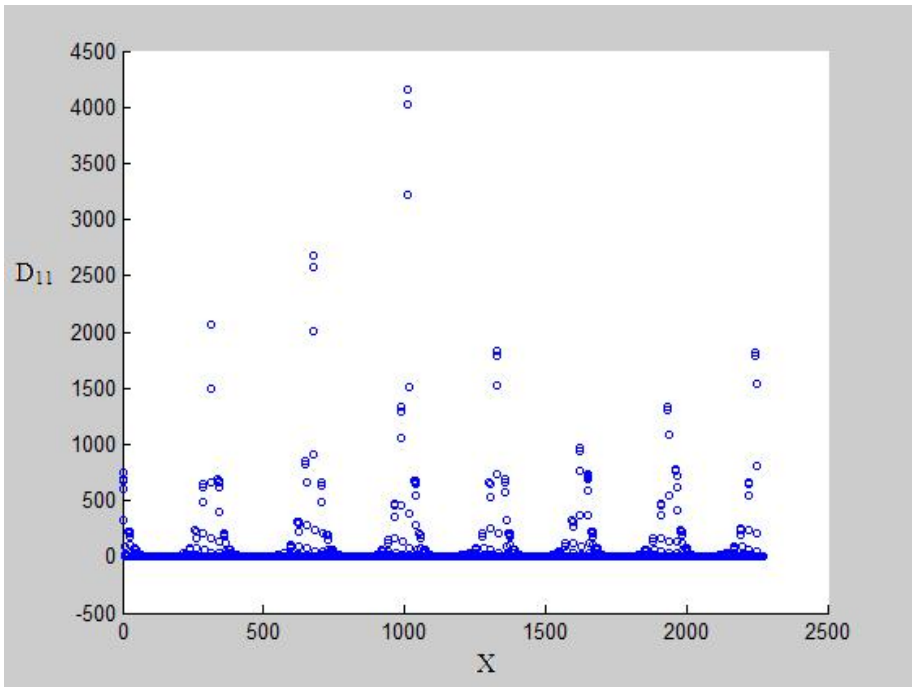

Ilustración 19

Valor mínimo = $-2{,}7690 \cdot 10^{-5}$. Valor máximo = 4154.

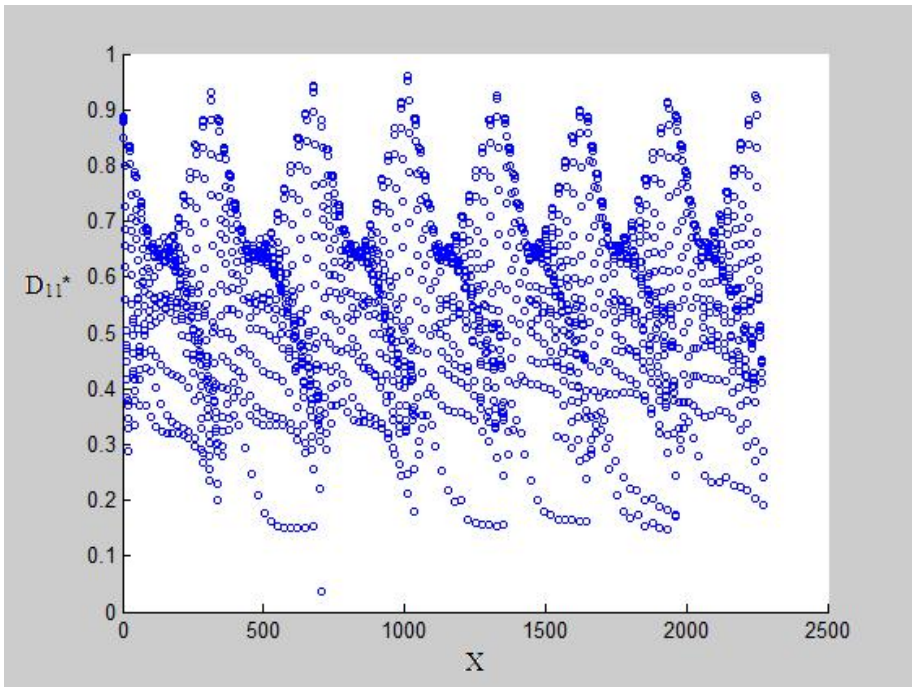

Ilustración 20

Valor mínimo = 0,0364. Valor máximo = 0,9618.

- Salida 2, *coeficiente monoenergético de conductividad* (D_{33}):

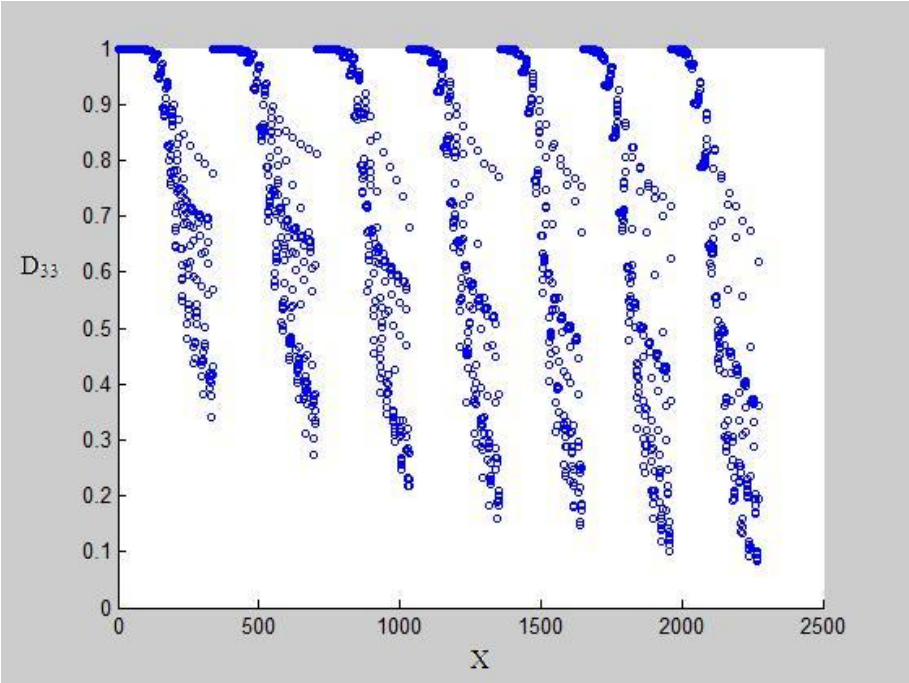

Ilustración 21
Valor mínimo = 0,0843. Valor máximo = 1.

Otro paso previo a entrenar la red fue el de repetir ciertos valores en la lista de datos de D_{33}. Como podemos observar, tras la normalización de D_{11}, los distintos valores quedan uniformemente repartidos por todo el intervalo. En el caso de D_{33} esto no es así, ya que un gran número de datos tienen por valor 1 o están muy cerca de él. Debido a esto y con el objetivo de dotar de relevancia al resto de los datos, reincluimos los valores menos *comunes* varias veces en la lista de datos. El número de veces que reincluimos estos valores se eligió viendo cómo variaba la velocidad de convergencia del proceso de entrenamiento de la red y los resultados que con esto se conseguían.

Pasamos ahora a *barajar* los datos. Con el objetivo de obtener una muestra suficientemente aleatoria del conjunto, barajamos todos los datos y luego cogimos un porcentaje de ellos, en nuestro caso el 70%. Este primer 70% lo utilizaremos para entrenar la red, empleando el 30% restante para testear cómo de buena es esta,

pidiéndole que obtenga esos resultados con los cuales no ha sido entrenada y comparándolos luego con los valores almacenados en nuestra base de datos.

El método utilizado para barajar los datos es el siguiente:

Siendo *N* el número total de datos (ya incluidos los repetidos en el caso de D_{11}), multiplicamos este por *rand* (número aleatorio entre 0 y 1) y utilizamos la función *floor* de MATLAB que nos deja la parte entera del número que se le introduce dentro. Vamos desde *i=1* hasta *N* multiplicando por *rand*, obteniendo así números aleatorios comprendidos entre 1 y *N*.

```
     % en el vector aux almacenaremos todos los valores que vaya
     % tomando i para compararlos y que no se repitan.
     aux=zeros(1,N)
for j=1:1:N;
     i=floor(rand.*N);
     % Introducimos un bucle para que mientras el valor de i sea cero,
     % o igual a uno obtenido anteriormente, el sistema obtenga otro
     % número que nos sirva
     aux(j)=i;
       while i==0 | i==aux(1,1:j)
        i=floor(rand.*N);
       end
     aux(j)=i;

     Ab(j)=An(i);
     Bb(j)=Bn(i);
     Cb(j)=Cn(i);
     Sb(j)=Sn(i);
end
```

Donde *A*, *B* y *C* son los vectores con los datos de las tres entradas del sistema y *S* es el vector con los datos de salida de la red. *b* indica que el vector está barajado y *n* indica normalizado.

Hecho esto, cogemos los n primeros datos de los vectores *Ab*, *Bb* y *Cb* siendo *n=floor(0,7*N)* y los agrupamos en la matriz *abc* de tres filas en el caso de D_{11} y *ab* de dos filas en el caso de D_{33} ya que para valores de $Er/v\,B$ bajo, este coeficiente no depende del valor de esta entrada. *abc* y *ab* constarán de *n* columnas. Creamos también así el vector "s", que contendrá los n primeros datos del vector salida. Habrá un vector *s*, ya sea D_{11} o D_{33}.

Con esto ya tenemos preparados los vectores de entrenamiento para la red.

Le decimos al programa cuantas redes queremos crear (cuantas más, más probabilidades hay de encontrar una satisfactoria) y este creará unas matrices multidimensionales donde se almacenan los distintos pesos obtenidos para cada red.

5.3 Creación de las redes

Nuestras redes constarán de una capa oculta y hemos usado dos funciones de transferencia distintas para ellas. Los datos de entrada a la capa oculta, que son los datos de entrada de la red, se tratan con la función "tansig", que no es otra que la famosa función tangente sigmoide. Los de salida de la capa oculta son tratados con "purelin", la cual es una simple transformación lineal, ya que como hemos comentado en el apartado *Backpropagation* del capítulo 2, el ajuste de estos pesos es más sencillo.

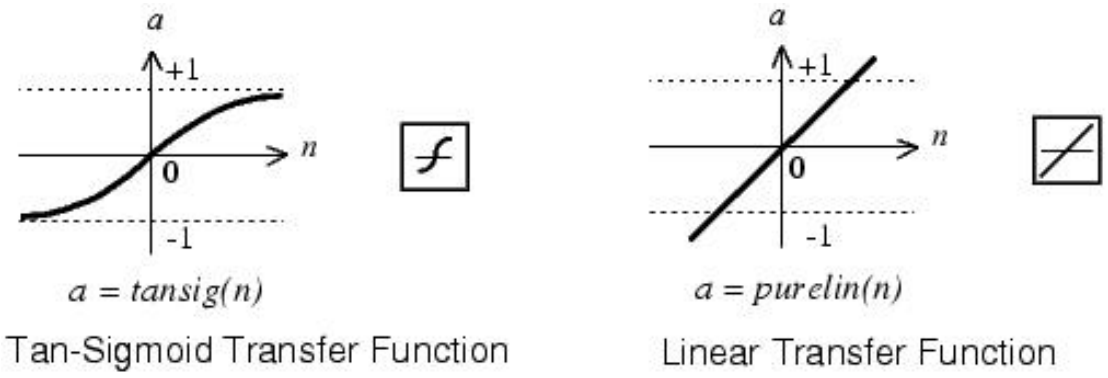

Ilustración 22

Como comentamos en apartados anteriores, el método de entrenamiento seguido es el de Levenberg-Marquardt. Este método es el que ofrece MATLAB por defecto ya que para redes con un número de pesos no demasiado elevado, es rápido y muy potente. Lo elegimos porque probamos con varios métodos y este fue el que obtuvo notablemente mejores resultados.

Por tanto, el código que crea la red es el siguiente:

```
net=newff(ab,s,nodos,{'tansig','purelin'},'trainlm');
```

Donde *newff* indica que la red a crear es una red *Feedforward*, que es el tipo de red más común a la hora de usar Backpropagation, y *nodos* es el número de nodos o

neuronas que le hemos dicho a MATLAB que queremos que tenga la red en la capa oculta.

El número de conexiones (que se traduce en el número de pesos que la red tiene que ajustar), vendrá dado por la siguiente ecuación con la ilustración 23 a modo de explicación:

$$N^{o}\,conexiones = N^{o}\,nodos \times N^{o}\,entradas + N^{o}\,nodos \times N^{o}\,salidas + nodos \times 1\;\; capa\;\; oculta + N^{o}\,salidas$$

(Ecuación 29)

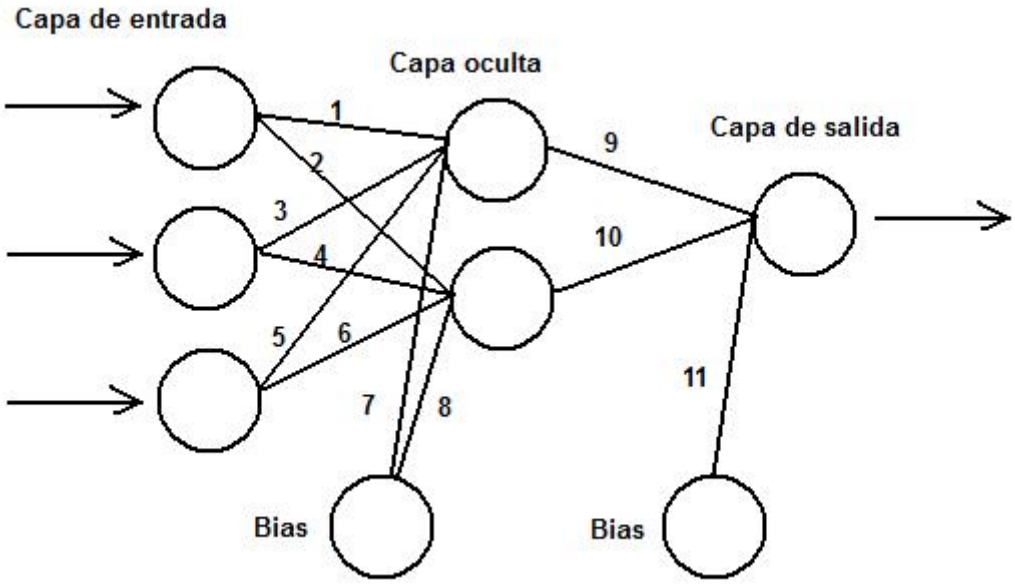

Ilustración 23

En este ejemplo, la red consta de 3 entradas, 2 nodos en la capa oculta y 1 salida.

Realizamos el cálculo:

$$N^{o}\,conexiones = 2 \times 3 + 2 \times 1 + 2 \times 1 + 1 = 11$$

Por lo tanto, esta red tendría 11 pesos.

Una vez creada la red, sólo queda entrenarla. Primero vamos a ajustar algunos parámetros:

- Para que no pare de iterar demasiado pronto, hemos cambiado el número máximo de iteraciones que no mejoran el resultado anterior de 6 a 15.

```
% Número máximo de iteraciones que no mejoran el resultado anterior
net.trainParam.max_fail=15;
```

- Para que la primera iteración sea una mera aproximación a un resultado bueno y no dure demasiado, hemos reducido las iteraciones máximas a 30 (luego lo aumentaremos). El valor del mínimo baja mucho de golpe y cortando las iteraciones pronto conseguimos que la red no se estanque buscando un buen resultado.

```
% Iteraciones durante las cuales busca el mínimo
net.trainParam.epochs = 30;
```

- Con todo ya preparado, entrenamos la red:

```
[net,tr] = train(net,ab,s);
```

A continuación se muestra una gráfica donde se ve cómo va reduciéndose el error con cada iteración:

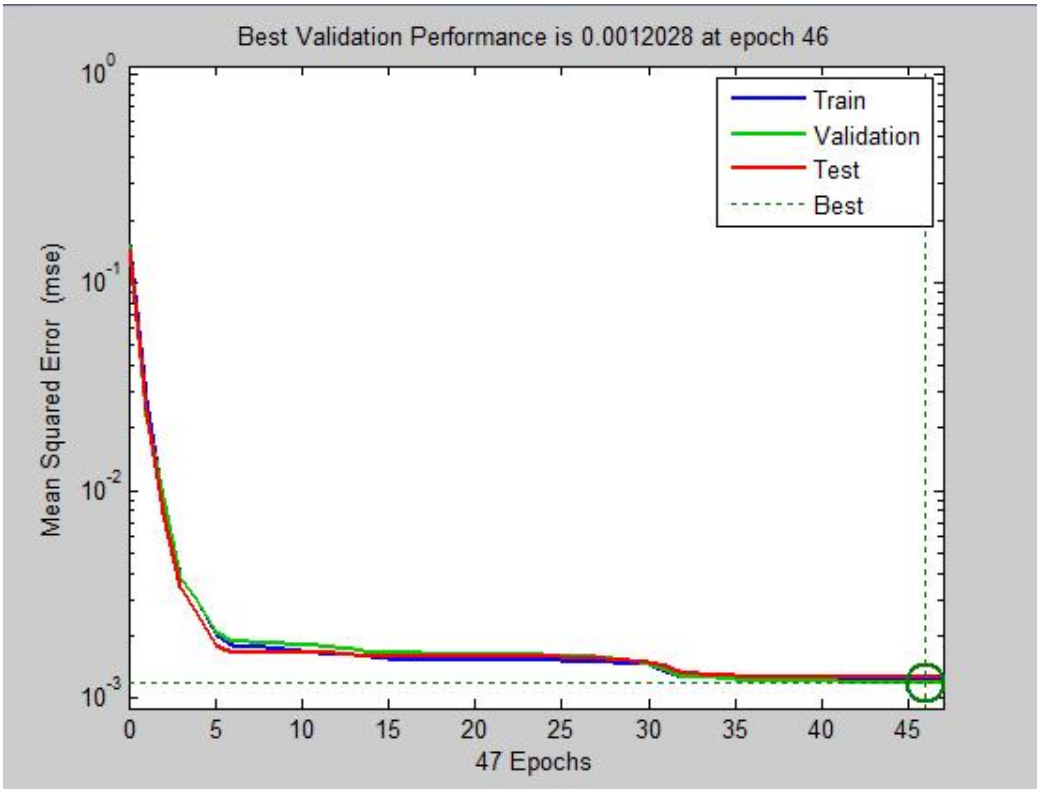

Ilustración 24

Lo que vamos a hacer a continuación es variar los valores de la matriz de pesos de entrada que sean demasiado grandes, puesto que esto nos indica que están saturados y no contribuyen al buen comportamiento de la red. Después someteremos a la red a otra sesión de entrenamiento partiendo ahora de otra posición en el espacio de parámetros al haber realizado los cambios citados. Al variar estos pesos, varían automáticamente los de las otras capas al reentrenar la red.

Lo primero que hacemos es aumentar el número de iteraciones máximas a 500, para darle tiempo al programa a que encuentre un buen resultado, luego variamos los pesos.

```
net.trainParam.epochs = 500;

for j=1:nodos

    % Variamos los pesos de la matriz de entrada que sean
    % excesivamente grandes

    net.IW{1,1}(j,1)=net.IW{1,1}(j,1)+(ALFA)*(rand-0.5);
    net.IW{1,1}(j,2)=net.IW{1,1}(j,2)+(ALFA)*(rand-0.5);

end
```

Donde net.IW es la matriz de los pesos de entrada a la red, ALFA es un factor que minimiza el incremento del peso, el cual está multiplicado por un valor aleatorio que varía entre -0,5 y 0,5 y viene dado por *(rand - 0,5)*.

Hemos diseñado el programa para que repita el proceso un número determinado de veces y que memorice los resultados que hayan encontrado el menor valor de error entre los datos y las salidas de la red. Tras dejar el programa iterando durante un tiempo, sólo queda comparar las redes obtenidas por este y elegir la que mejor describa los datos.

- *Selección de la red entre las obtenidas:*

Para seleccionarla nos basaremos en dos criterios:

- *Criterio nº 1, criterio visual (cualitativo):*

  Introducimos las entradas del 30 % de los datos sobrantes que quedaban, haciendo que la red obtenga las salidas. Tras esto, enfrentaremos los datos obtenidos por la red con los que ya teníamos. El resultado perfecto sería obtener una serie de puntos justo en de la recta $y = x$.

Como ejemplo mostramos a continuación la comparativa de dos gráficas obtenidas para D_{11}, donde vemos que el segundo resultado es mejor que el primero.

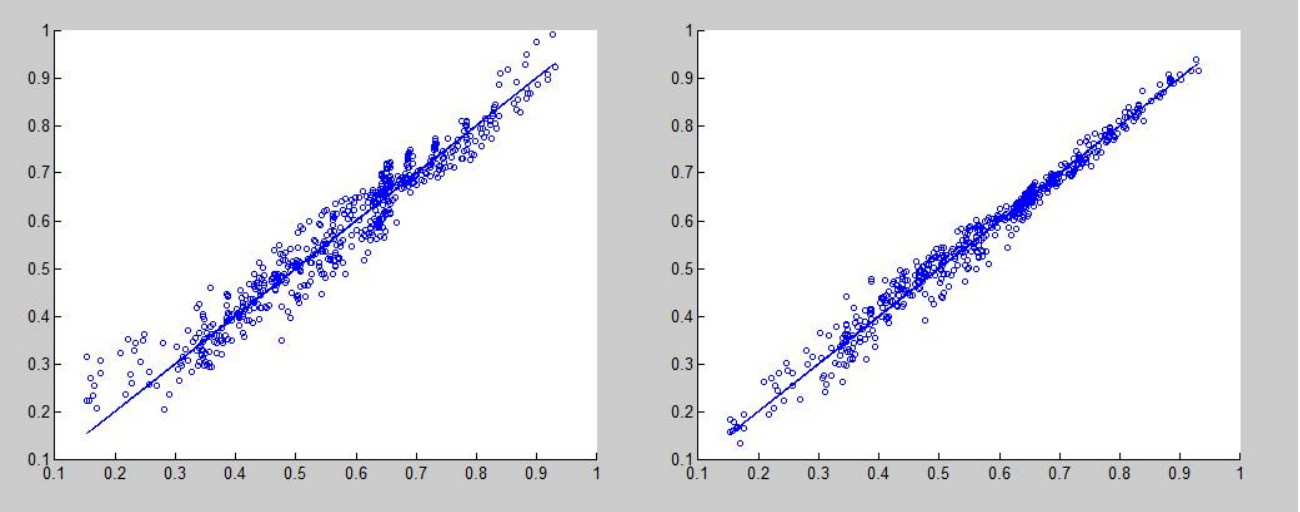

Ilustración 25

- *Criterio nº 2, criterio basado en el error numérico (cuantitativo):*

  Una vez descartadas las redes claramente menos optimizadas siguiendo el primer criterio, comparamos el error medio numérico, así como la mediana y el error máximo de las redes que habían pasado la primera criba, elegimos la que a nuestro parecer es la más óptima.

El último paso que queda por dar tras esto es el de preparar al programa para desnormalizar los datos a la salida de la red y esta queda lista para funcionar.

6 RESULTADOS OBTENIDOS

El objetivo de este capítulo es mostrar los resultados obtenidos para cada uno de los coeficientes a ajustar. Mostraremos estos resultados a través de varias gráficas que facilitarán la comprensión de los mismos.

6.1 Red de coeficientes D_{11}

La red finalmente elegida consta de 8 nodos, por lo tanto la red tendrá que ajustar 41 pesos. En un principio habíamos elegido una red de 4 nodos, reduciéndose bastante el número de pesos a ajustar (21) y por lo tanto reduciendo los cálculos a realizar por la red para obtener resultados. Pese a que 41 es un número considerable de conexiones, la velocidad de trabajo de la red es muy elevada y consideramos que hoy en día cualquier ordenador corriente puede hacerla funcionar en perfectas condiciones.

Concretamente, mi PC (doble núcleo de 2,67GHz) tardó 0,011778 segundos en calcular y desnormalizar el 30 % de los datos que habíamos dejado para testear con la red de 4 nodos, mientras que con la de 8 tardó 0,012331 segundos. La diferencia como observamos es insignificante frente a la mejora que supone una red frente a la otra.

En la gráfica de la ilustración 26 aparecen representados los D_{11} para $r/a=0,5$ y los mismos valores de $Er/(vB)$ que los representados en la ilustración 12 de la página 38. Las líneas continuas han sido dibujadas con los valores obtenidos por la RNA, mientras que los puntos son los valores de la base de datos con los que ha sido entrenada la red, obtenidos con el método MOCA. Tras esta, mostramos una gráfica con los datos obtenidos por la red enfrentados con los obtenidos con el método teórico.

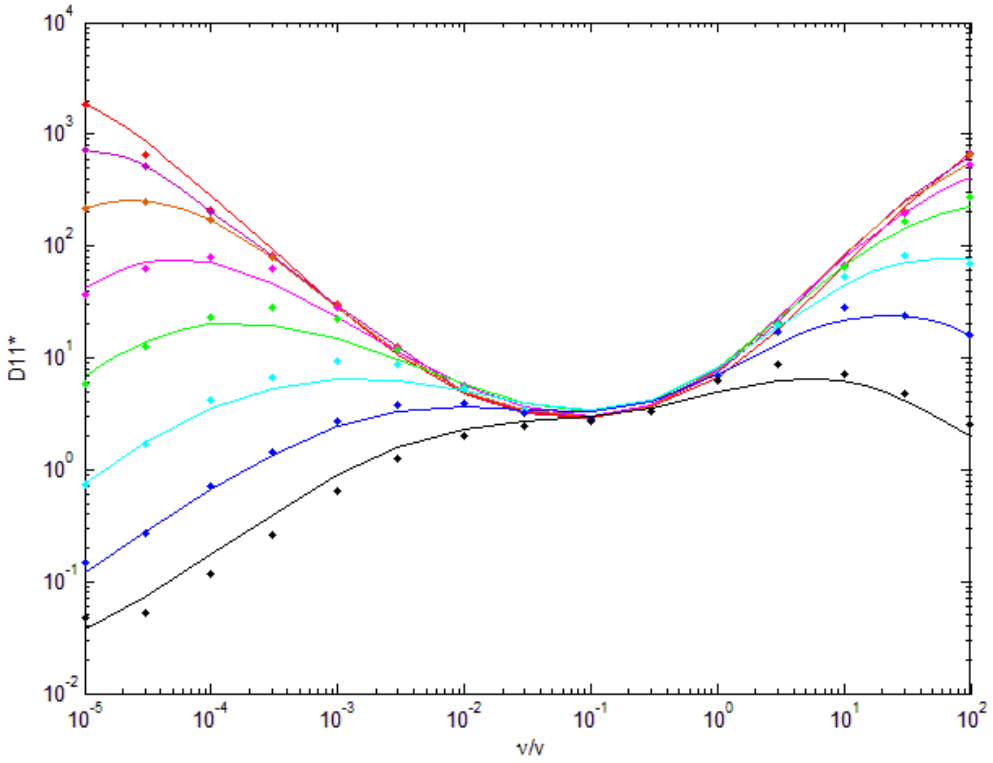

Ilustración 26
Er/vB = 0 [rojo], 1x10-5 [púrpura], 3x10-5 [naranja], 1x10-4[magenta], 3x10-4 [verde], 1x10-3 [azul claro], 3x10-3[azul], 1x10-2 [negro]. Líneas obtenidas con RNA D11. Puntos obtenidos con el método MOCA.

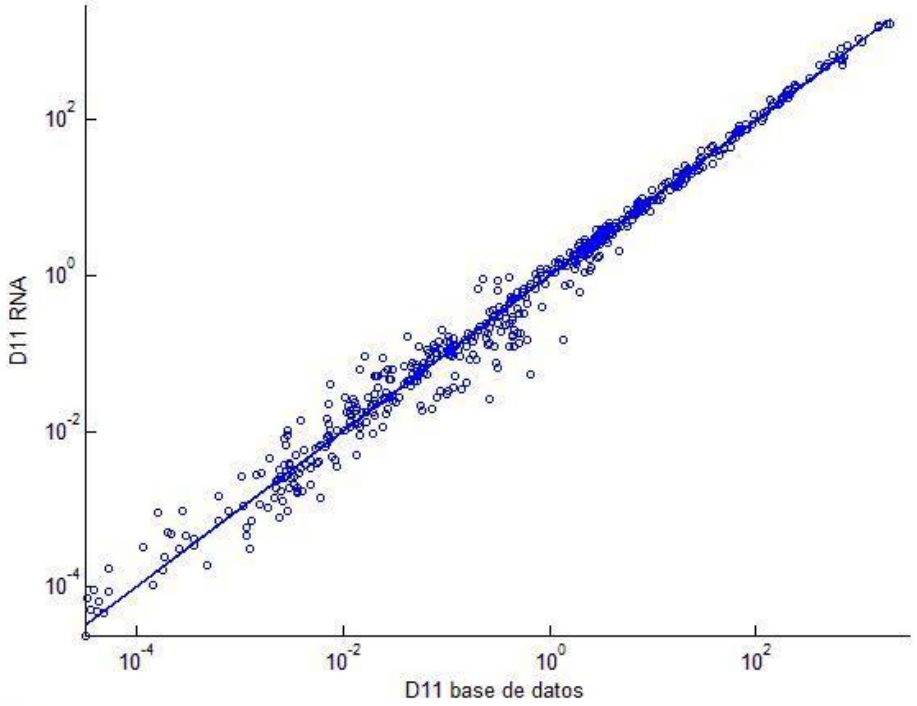

Ilustración 27

Errores de D_{11}		
ErrorMedio = 4’ 7404	ErrorMediana = 0’1135	ErrorMax = 275’9790

- *Conclusiones sobre la red D_{11}:*

A la vista de los resultados obtenidos, podemos asumir que la gran mayoría de los errores están muy alejados del máximo cuyo valor es 275’979 el cual no es un valor demasiado grande teniendo en cuenta que estamos tratando con valores que llegan hasta 3000 y 4000. Los valores de la mediana y el error medio nos dan una idea de cómo de grande son los errores y la cantidad de estos que se acercan al mínimo y al máximo. La mediana nos dice que la mitad de los errores están por debajo de 0’1135, el cual es un valor bastante pequeño. El error medio cuyo valor es 4’7404 nos hace ver que pese a haber valores relativamente grandes como 275’979 casi todos son mucho más pequeños, lo que hace que este se reduzca tanto.

Los resultados por tanto son satisfactorios ya que se ajustan en gran medida a los valores teóricos y la velocidad con la que han sido obtenidos ha sido muy elevada, ya que una vez implementada la red, esta sólo realiza operaciones elementales. Hemos de tener en cuenta no obstante que no hubiéramos podido entrenar la red sin dedicar las 50000 horas de trabajo previo para obtener la base de datos, pero una vez creada nuestra RNA es una potente herramienta que nos proporcionará los resultados deseados casi de inmediato y con un alto grado de fiabilidad.

6.2 Red de coeficientes D_{33}

La red elegida consta de 3 nodos y por tanto de 13 pesos. Es un número considerablemente más bajo que el escogido para la anterior red, pero hemos de tener en cuenta que en esta sólo habrá dos entradas, por lo que será una red relativamente más sencilla de ajustar.

Cómo para D_{11}, comparamos los datos de D_{33} de la base de datos con los obtenidos por la red para los valores de *r/a* y *Er/(vB)* de las gráficas de la ilustración 12 (página 38). En la ilustración 29 mostramos la gráfica con los datos obtenidos por la red enfrentados con los obtenidos con el método teórico DKES.

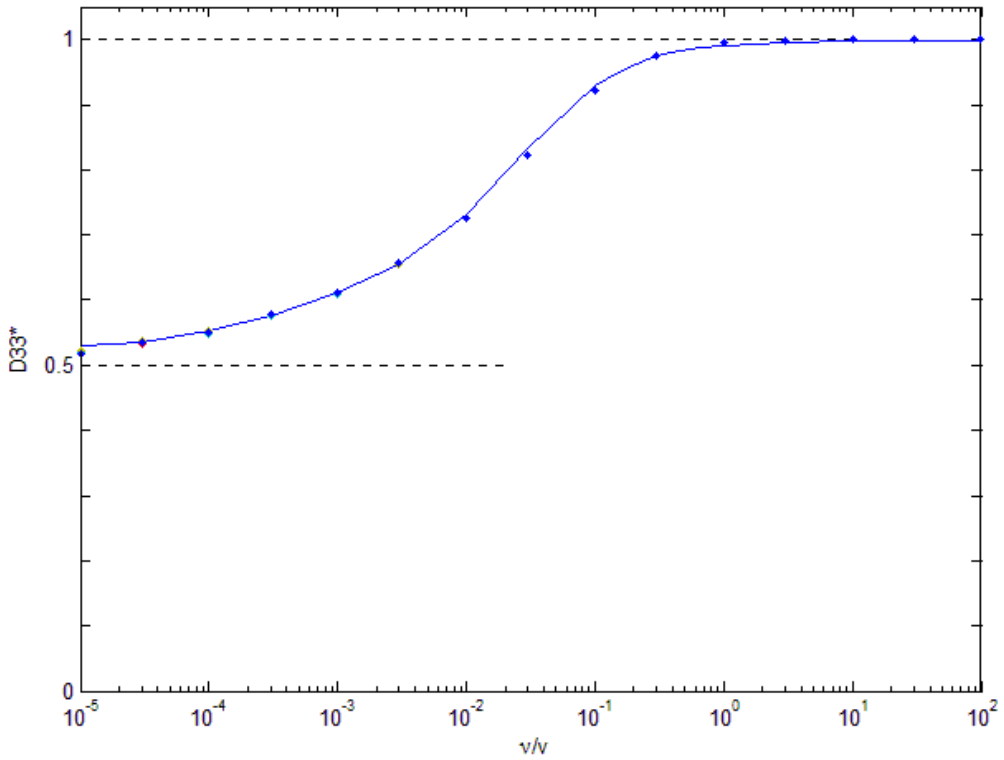

Ilustración 28
Er/vB = 0 [rojo], 1 x 10-5 [púrpura], 3x10-5 [naranja], 1x10-4[magenta], 3x10-4 [verde], 1x10-3 [azul claro], 3x10-3[azul], 1x10-2 [negro]. Líneas obtenidas con RNA D33.
Puntos obtenidos con el método DKES.

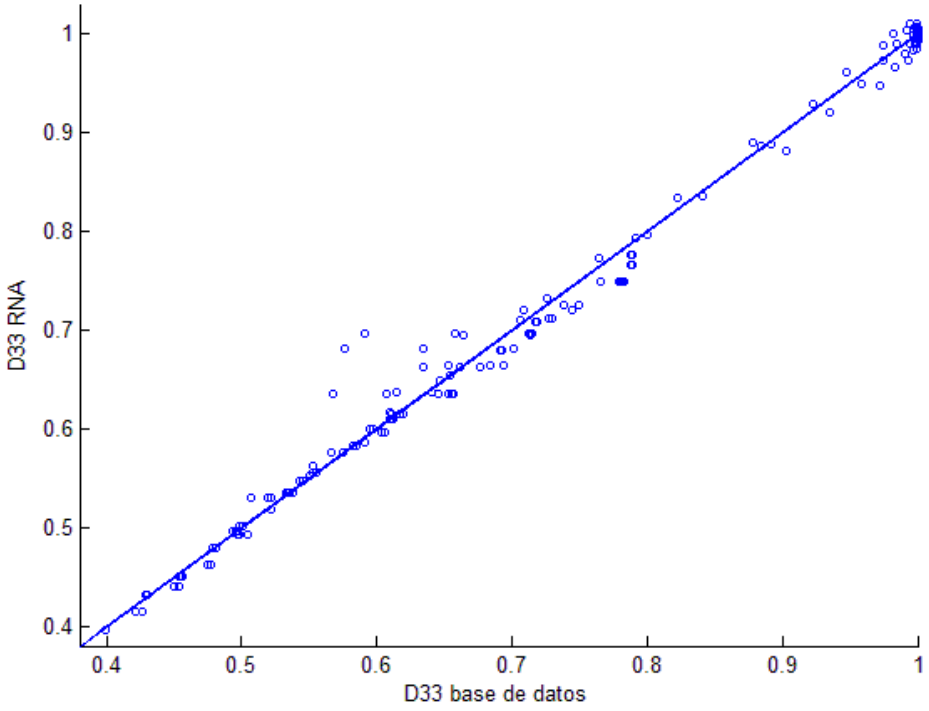

Ilustración 29

Errores absolutos de D_{33}		
ErrorMedio = 0’0103	ErrorMediana = 0’0064	ErrorMax = 0’1053

- *Conclusiones sobre la red D_{33}:*

Podemos observar en la gráfica que la gran mayoría de los datos *caen* sobre la recta $y=x$, alejándose de ella sólo unos pocos una distancia no demasiado grande. Por otra parte, casi todos los datos se encuentran por debajo del 2 % del error (si consideramos 1 como el 100%) y más de la mitad están por debajo del 1%, sólo dos datos se alejan mucho de esto llegando aproximadamente al 10% de error.

Los resultados son muy satisfactorios pese a los dos puntos que se alejan ya que los demás están muy por debajo, habiendo un gran número de ellos que no pasan del 1%. Recordamos no obstante lo mencionado en las conclusiones de D_{11}, ya que sin la base de datos y por lo tanto sin emplear todo ese tiempo en conseguirla, no podríamos haber creado esta RNA que ahora es una herramienta de trabajo potente y fiable para conseguir datos de D_{33} comprendidos dentro los márgenes entre los que se mueven los valores de este coeficiente en la base de datos.

7 POSIBLES APLICACIONES DE LAS REDES NEURONALES

La posible aplicación que proponemos es la de utilizar un sistema de redes neuronales que regule la climatización de un espacio cerrado, ya sea un vehículo, una habitación o una planta entera de un edificio. Para explicar nuestra propuesta usaremos un ejemplo conciso, aunque perfectamente extrapolable a las situaciones citadas anteriormente.

Muchas empresas alquilan plantas enteras para usar como oficina general de uso común por todos sus empleados. A menudo estos sistemas de refrigeración funcionan con grandes bombas de frío-calor que regulan la temperatura de una o varias plantas. Estos sistemas suelen ser ineficientes, ya que los termostatos controlan sólo la temperatura en un punto y muchos no tienen en cuenta las diferencias de temperatura en las distintas zonas de la planta en cuestión debido al flujo de calor que escapa a través de las ventanas en invierno o del calor que genera el sol en verano en estas zonas de la oficina.

Los datos que estos sistemas suelen pedir al usuario dependen del volumen a refrigerar. Las variables que normalmente piden los sistemas que regulan la temperatura en casos como el que hemos elegido son la presión y la temperatura de salida de la bomba. Pues bien, nuestro sistema dependerá de unas cuantas variables más.

A continuación pasamos a describir el ejemplo a tratar. La planta de oficinas consta de:

- Una bomba de frío-calor con capacidad suficiente para aumentar o disminuir la temperatura hasta valores de confort aceptables de todo el volumen de aire.
- Tres salidas de aire en el techo con apertura regulable.
- Cinco sensores de temperatura distribuidos estratégicamente.

La ilustración que sigue muestra un pequeño esquema de la planta. Los sensores de temperatura se muestran como puntos en rojo y las salidas de aire son los rectángulos de color naranja.

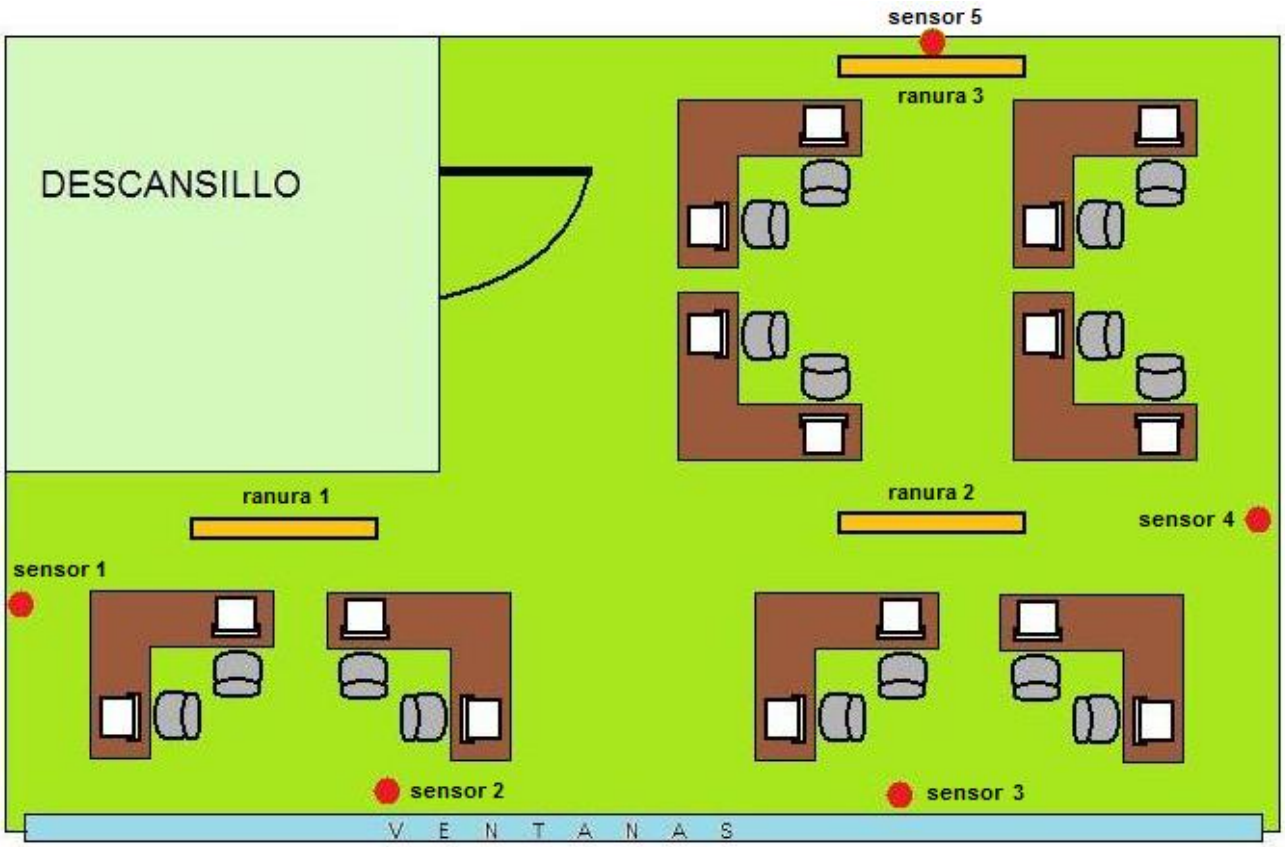

Ilustración 30

Entradas del sistema:

- Temperatura medida por cada sensor.
- Grado de apertura de cada compuerta.

Salidas del sistema:

- Presión de salida de la bomba de aire.
- Temperatura de salida de la bomba de aire.
- Grado de apertura de cada compuerta.

Como podemos observar, el grado de apertura de las distintas compuertas serán entradas y a la vez salidas de nuestro sistema. Esto quiere decir que el sistema se irá actualizando constantemente con los valores que él mismo obtendrá.

Crearemos una red para cada salida, mejorando así el óptimo funcionamiento del sistema ya que un problema con tantas salidas es mucho más complicado de resolver que cinco problemas con una salida cada uno. Por lo tanto, para nuestro caso particular crearemos cinco RNA de 8 entradas y 1 salida cada una que pasamos a describir en el siguiente apartado.

7.1 Redes del sistema

RNA nº 1:

- Entradas:

  Sensor 1, sensor 2, sensor 3, sensor 4, sensor 5, grado apertura ranura 1, grado apertura ranura 2, grado apertura ranura 3.

- Salidas:

  Grado de apertura ranura 1.

RNA nº 2:

- Entradas:

  Sensor 1, sensor 2, sensor 3, sensor 4, sensor 5, grado apertura ranura 1, grado apertura ranura 2, grado apertura ranura 3.

- Salidas:

  Grado de apertura ranura 2.

RNA nº 3:

- Entradas:

  Sensor 1, sensor 2, sensor 3, sensor 4, sensor 5, grado apertura ranura 1, grado apertura ranura 2, grado apertura ranura 3.

- Salidas:

  Grado de apertura ranura 3.

RNA nº 4:

- Entradas:

  Sensor 1, sensor 2, sensor 3, sensor 4, sensor 5, grado apertura ranura 1, grado apertura ranura 2, grado apertura ranura 3.

- Salidas:

Presión de salida de la bomba de aire.

RNA nº 5:

- Entradas:

  Sensor 1, sensor 2, sensor 3, sensor 4, sensor 5, grado apertura ranura 1, grado apertura ranura 2, grado apertura ranura 3.

- Salidas:

  Temperatura de salida de la bomba de aire.

7.2 Proceso de entrenamiento del sistema

Como vemos tenemos dos redes muy diferenciadas de las otras tres, pero esto no afectará en absoluto al entrenamiento del sistema, que será completamente individualizado para cada una de sus partes. Una vez decidido el mayor número de nodos necesarios es posible manufacturar cada una de las redes neuronales con DSP (Procesadores Digitales de Señales) que pueden ser programados remotamente con los valores de los pesos, por lo tanto **no es necesario tener un ordenador que se encargue de esto en cada oficina.**

Necesitamos una o varias jornadas de toma de datos. En este periodo, se variarán constantemente los parámetros de entrada tomando nota de los de salida. Para una buena obtención de los datos, necesitamos dejar un pequeño periodo de tiempo tras cada variación de parámetros para llegar a una temperatura estable y uniforme en la planta.

Una vez terminado el periodo de entrenamiento, la idea es marcar una única temperatura en toda la sala y que el sistema sea capaz de mantenerla lo mejor posible, es decir que los cinco sensores marquen aproximadamente la misma temperatura.

Será necesario obtener un rango de datos lo suficientemente amplio en función de las necesidades que tengamos. Para un óptimo funcionamiento de nuestro sistema, sería

necesario obtener datos en distintas épocas del año, en días extremadamente calurosos, en días fríos y en días no tan extremos.

Pese a que esto pueda parecer un gran impedimento si tenemos una planta que climatizar y no queremos esperar a que pasen todas las estaciones, esto es solventable si actualizamos cada cierto tiempo la red con nuevos valores. Podemos hacer un entrenamiento rápido que consiga obtener suficientes datos sobre las condiciones actuales del clima para poner en marcha nuestro sistema, y cada cierto tiempo, tomar nuevos datos y reentrenar las redes existentes con los que ya teníamos y estos nuevos. Esto es relativamente sencillo, basta con añadir estos datos a la lista de los que ya habíamos usado para el aprendizaje de la red y volver a realizar un entrenamiento. Antes de esto es aconsejable ver si nuestra red es capaz de predecir las salidas correctamente con los nuevos datos de entrada, en este caso no haría falta reentrenarla. Esto lo haremos con los métodos utilizados para analizar los resultados obtenidos en el problema tratado en el presente proyecto: el método visual en el que enfrentamos valores en gráficas y midiendo el error como la diferencia entre puntos obtenidos por la toma de datos y puntos obtenidos por las distintas redes neuronales que componen nuestro sistema. Con este método conseguiremos un sistema suficientemente efectivo que se irá mejorando con el transcurso del tiempo. Una vez pasado un periodo aproximado de un año podremos decir que el sistema está lo suficientemente preparado para no tener que volver a ser reentrenado.

Por último, podemos incluir una pequeña unidad, con el equivalente a un teléfono móvil, que reenvíe los datos de vez en cuando a nuestra oficina central para recalcular, y así poder recargar los nuevos pesos o las nuevas actualizaciones de software.

APÉNDICE A

PROGRAMAS PARA D_{11}

PROGRAMA "CrearRedesD11"

```
% Pedimos los datos de entrada
nodos=input('Número de nodos: ');
NUM=input('Número de redes a crear: ');

% Creamos las matrices multidimensionales que almacenarán los pesos de
las mejores redes obtenidas
IW=zeros(nodos,3,NUM);
LW=zeros(1,nodos,NUM);
bias=zeros(nodos,1,NUM);
bias2=zeros(1,1,NUM);
PERFORMANCE=zeros(1,NUM);

& Para crear por primera vez los datos, creamos la muestra y la
preparamos para entrenar la red con la subrutina "crear".
Guardamos los datos y para una próxima vez, los cargamos con la
subrutina "cargardatos"
run crear11
```

Subrutina "crear11"

```
% Cargamos los datos de entrada "Inputs" y los de salida "Targets"
load Inputs; load Targets;

% Almacenamos los datos en vectores separados
N=2272; A=Inputs(1,1:1:N); B=Inputs(2,1:1:N); S=Targets(1,1:1:N);
C=Inputs(3,1:1:N);

% Normalizamos los datos
An=A; Bn=(log10(B)+6)/8; Cn=(log10(C+1e-6)+6)/7; Sn=S;

% Barajamos y obtenemos "abc" y "s" con la subrutina "tratdatos"
run tratdatos11
errormedio=1; errormax=1; inic=200; X=1:N;

% Guardamos los datos obtenidos para tratar siempre los mismos
save datos11
```

Subrutina "cargardatos11"

```
load datos11
```

Subrutina "tratdatos11"

```
%barajamos los datos:
%NOTA: Db va a ser una especie de guía, para no perder la posición de
los datos tras barajar
Ab=zeros(1,N); Bb=zeros(1,N);  Sb=zeros(1,N); Db=zeros(1,N);
Cb=zeros(1,N);

for j=1:1:N;
     i=floor(rand.*N);
     % Introducimos un bucle para que mientras el valor de i sea cero,
     % o igual a uno obtenido anteriormente, el sistema obtenga otro
     % número que nos sirva
     aux(j)=i;
       while i==0 | i==aux(1,1:j)
        i=floor(rand.*N);
       end
     aux(j)=i;

     Ab(j)=An(i);
     Bb(j)=Bn(i);
     Cb(j)=Cn(i);
     Db(j)=Cn(i);
     Sb(j)=Sn(i);
end
P=0.7; n=floor(P*N); abc(1,1:1:n)=Ab(1:1:n); abc(2,1:1:n)=Bb(1:1:n);
abc(3,1:1:n)=Cb(1:1:n); s=Sb(1:1:n);
```

```
% Ponemos en marcha la creación de las redes
for contador=1:NUM

     % Corremos la subrutina "crearred" NUM veces
     run crearred11

    % Metemos los datos en matrices de 3 "cajones"
    IW(1:nodos,1:3,contador)=PESOSent;
    LW(1,1:nodos,contador)=PESOSsal;
    bias(1:nodos,1,contador)=PESOSfijos;
    bias2(1,1,contador)=PESOSfijos2;
    PERFORMANCE(1,contador)=perf;
end
```

Subrutina "crearred11"

```
% Coeficiente de variación de los pesos saturados ALFA
ALFA=0.2;

% Creamos la red
net=newff(abc,s,nodos,{'tansig','purelin'},'trainlm');

net.trainParam.epochs = 30;
net.trainParam.show = 50;

% Entrenamos la red
[net,tr] = train(net,abc,s);

% Variamos los pesos de la red con la subrutina "variarpesos"
run variapesos11
% Borramos la red obtenida para crear una nueva
clear net
```

Subrutina "variarpesos11"

```
% Reseteamos los datos
aux=1; tr.perf(aux)=1; MEJORperf=1; i=0; k=0; z=0; LIMITE1=2.5;
LIMITE2=1.1;

% Se define el número de veces que la red va a modificar los pesos y
se corre el subprograma
ite=100;
     for i=1:ite
        net.trainParam.epochs = 500;
        for j=1:nodos

        % Variamos los pesos de la matriz de entrada que sean
        excesivamente grandes

              if abs(net.IW{1,1}(j,1))>LIMITE1
                  net.IW{1,1}(j,1)=net.IW{1,1}(j,1)/1.5;
              elseif abs(net.IW{1,1}(j,1))>LIMITE2
                  net.IW{1,1}(j,1)=net.IW{1,1}(j,1)+(ALFA)*(rand-0.5);
              end

              if abs(net.IW{1,1}(j,2))>LIMITE1
                  net.IW{1,1}(j,2)=net.IW{1,1}(j,2)/1.5;
              elseif abs(net.IW{1,1}(j,2))>LIMITE2
                  net.IW{1,1}(j,2)=net.IW{1,1}(j,2)+(ALFA)*(rand-0.5);
              end

              if abs(net.IW{1,1}(j,3))>LIMITE1
                  net.IW{1,1}(j,3)=net.IW{1,1}(j,3)/1.5;
```

```
            elseif abs(net.IW{1,1}(j,3))>LIMITE2
                net.IW{1,1}(j,3)=net.IW{1,1}(j,3)+(ALFA)*(rand-0.5);
            end

        end

            % Reentrenamos la red partiendo de los pesos modificados
            [net,tr]=train(net,abc,s);
            aux=length(tr.perf);

            % Guardamos los mejores resultados en función del perf
            (mínimo más pequeño que ha encontrado el train)
            if tr.perf(aux)<MEJORperf
                MEJORperf=tr.perf(aux);
                PESOSent=net.IW{1,1};
                PESOSsal=net.LW{2,1};
                PESOSfijos=net.b{1,1};
                PESOSfijos2=net.b{2,1};
                perf=tr.perf(aux);
            end

    end
```

```
% Guardamos los resultados finales en "REDESobtenidas"
save REDESobtenidas11 IW LW bias bias2 PERFORMANCE nodos
```

FIN PROGRAMA "CrearRedesD11"

PROGRAMA "VERRED11"

```
% Elegimos el número de la red que queremos ver (están numeradas por
orden de creación)
run cargardatos11
load REDESobtenidas11
NUM=input('Número de la red a crear: ');
net=newff(abc,s,nodos);
net.IW{1,1}=IW(:,:,NUM);
net.LW{2,1}=LW(1,:,NUM);
net.b{1,1}=bias(:,1,NUM);
net.b{2,1}=bias2(1,1,NUM);

% Creamos los datos que habíamos dejado para testear con "creardatos"
y los enfrentamos con los valores teóricos
run creardatos11
scatter(Srec,Sff,2.3); hold on; rectaux=Srec; plot(Srec,rectaux);
performance=PERFORMANCE(1,NUM)
```

Subrutina "creardatos11"

```
% Creamos los datos
SFFn=sim(net,[Ab; Bb; Cb]);

% Desnormalizamos los datos creados
SFF=10.^((SFFn-0.6).*10)-1e-6;

% Seleccionamos los datos con los que no ha sido entrenada la red
Sff=SFF(n+1:1:N);

% Creamos un vector con los datos teóricos en el mismo orden que los
datos barajados, para enfrentar cada uno con el que corresponda
Srec=S(Db(n+1:1:N));
```

FIN PROGRAMA "VERRED11"

PROGRAMA "ERROR11"

```
% Una vez vistos los datos creados por la red enfrentados con los
teóricos, vamos a ver el error medio, la mediana y el error máximo con
este programa
% Creamos el vector Verror donde se almacenan los errores en valor
absoluto dato por dato y el vector Verror2, donde se almacenan los
errores con signo, para enfrentarlos luego con el vector X que va
desde 1 hasta N-n
Verror=zeros(1,N-n);
Verror2=zeros(1,N-n);

run creardatos11
for i=1:N-n
    Verror(i)=abs(Sb(i+n)-Sff(i));
    Verror2(i)=Sb(i+n)-Sff(i);
end
ErrorMedio=mean(Verror)
ErrorMax=max(Verror)

% Creamos el vector X y lo enfrentamos con Verror2
X=1:1:N-n;
scatter(X,Verror2,2.3); hold on; plot(X,0);
```

FIN PROGRAMA "ERROR11"

APÉNDICE B

PROGRAMAS PARA D_{33}

PROGRAMA "CrearRedesD33"

```
% Pedimos los datos de entrada
nodos=input('Número de nodos: ');
NUM=input('Número de redes a crear: ');

% Creamos las matrices multidimensionales que almacenarán los pesos de
las mejores redes obtenidas
IW=zeros(nodos,3,NUM);
LW=zeros(1,nodos,NUM);
bias=zeros(nodos,1,NUM);
bias2=zeros(1,1,NUM);
PERFORMANCE=zeros(1,NUM);

& Para crear por primera vez los datos, creamos la muestra y la
preparamos para entrenar la red con la subrutina "crear33".
Guardamos los datos.
run crear33
```

Subrutina "crear33"

```
% Cargamos los datos de entrada "Inputs" y los de salida "Targets"
load Inputs; load Targets;

% Almacenamos los datos en vectores separados
N=2272; A=Inputs(1,1:1:N); B=Inputs(2,1:1:N); S=Targets(3,1:1:N);
C=Inputs(3,1:1:N);

% Normalizamos los datos
An=A; Bn=(log10(B)+6)/8; Cn=(log10(C+1e-6)+6)/7; Sn=S;

% Barajamos y obtenemos "ab" y "s" con la subrutina "tratdatos33"
run tratdatos33
errormedio=1; errormax=1; inic=200; X=1:N;

% Guardamos los datos obtenidos para tratar siempre los mismos
save datos33
```

```
% Una vez creados, lanzamos la subrutina “cargardatos33”, que carga
los datos y repite varias veces ciertos valores seleccionados adrede
con la intención de darles más peso en la red del que en principio,
por ser pocos, tendrían.
run cargardatos33
```

Subrutina “cargardatos33”

```
load datos33

for i=1:1:n
    if s(i)>0.566 && s(i)<0.8
        j=j+1;

        s(n+j)=s(i);
        ab(1,n+j)=ab(1,i);
        ab(2,n+j)=ab(2,i);

        s(n+j+1)=s(i);
        ab(1,n+j+1)=ab(1,i);
        ab(2,n+j+1)=ab(2,i);

        s(n+j+2)=s(i);
        ab(1,n+j+2)=ab(1,i);
        ab(2,n+j+2)=ab(2,i);

        s(n+j+3)=s(i);
        ab(1,n+j+3)=ab(1,i);
        ab(2,n+j+3)=ab(2,i);

        s(n+j+4)=s(i);
        ab(1,n+j+4)=ab(1,i);
        ab(2,n+j+4)=ab(2,i);

        s(n+j+5)=s(i);
        ab(1,n+j+5)=ab(1,i);
        ab(2,n+j+5)=ab(2,i);

        s(n+j+6)=s(i);
        ab(1,n+j+6)=ab(1,i);
        ab(2,n+j+6)=ab(2,i);

        j=j+6;
        z=z+1;
    end
end
numero=z;
```

Subrutina "tratdatos33"

```
%barajamos los datos:
%NOTA: Db va a ser una especie de guía, para no perder la posición de
los datos tras barajar
Ab=zeros(1,N); Bb=zeros(1,N);  Sb=zeros(1,N); Db=zeros(1,N);

for j=1:1:N;
     i=floor(rand.*N);
     % Introducimos un bucle para que mientras el valor de i sea cero,
     % o igual a uno obtenido anteriormente, el sistema obtenga otro
     % número que nos sirva
     aux(j)=i;
       while i==0 | i==aux(1,1:j)
        i=floor(rand.*N);
       end
     aux(j)=i;

     Ab(j)=An(i);
     Bb(j)=Bn(i);
     Db(j)=Cn(i);
     Sb(j)=Sn(i);
end
P=0.7; n=floor(P*N); ab(1,1:1:n)=Ab(1:1:n); ab(2,1:1:n)=Bb(1:1:n);
s=Sb(1:1:n);
```

```
% Ponemos en marcha la creación de las redes
for contador=1:NUM

     % Corremos la subrutina "crearred" NUM veces
     run crearred33

    % Metemos los datos en matrices de 2 "cajones"
    IW(1:nodos,1:2,contador)=PESOSent;
    LW(1,1:nodos,contador)=PESOSsal;
    bias(1:nodos,1,contador)=PESOSfijos;
    bias2(1,1,contador)=PESOSfijos2;
    PERFORMANCE(1,contador)=perf;
end
```

Subrutina "crearred33"

```
% Coeficiente de variación de los pesos saturados ALFA
ALFA=0.2;

% Creamos la red
net=newff(ab,s,nodos,{'tansig','purelin'},'trainlm');

net.trainParam.epochs = 30;
net.trainParam.show = 50;
```

```
% Entrenamos la red
[net,tr] = train(net,ab,s);

% Variamos los pesos de la red con la subrutina "variarpesos"
run variapesos33

% Borramos la red obtenida para crear una nueva
clear net
```

Subrutina "variarpesos33"

```
% Reseteamos los datos
aux=1; tr.perf(aux)=1; MEJORperf=1; i=0; k=0; z=0; LIMITE1=2;
LIMITE2=1;

% Se define el número de veces que la red va a modificar los pesos y
se corre el subprograma
ite=100;
     for i=1:ite
        net.trainParam.epochs = 500;
        for j=1:nodos

        % Variamos los pesos de la matriz de entrada que sean
        excesivamente grandes

              if abs(net.IW{1,1}(j,1))>LIMITE1
                  net.IW{1,1}(j,1)=net.IW{1,1}(j,1)/1.5;
              elseif abs(net.IW{1,1}(j,1))>LIMITE2
                  net.IW{1,1}(j,1)=net.IW{1,1}(j,1)+(ALFA)*(rand-0.5);
              end

              if abs(net.IW{1,1}(j,2))>LIMITE1
                  net.IW{1,1}(j,2)=net.IW{1,1}(j,2)/1.5;
              elseif abs(net.IW{1,1}(j,2))>LIMITE2
                  net.IW{1,1}(j,2)=net.IW{1,1}(j,2)+(ALFA)*(rand-0.5);
              end

        end

             % Reentrenamos la red partiendo de los pesos modificados
             [net,tr]=train(net,ab,s);
             aux=length(tr.perf);

             % Guardamos los mejores resultados en función del perf
             (mínimo más pequeño que ha encontrado el train)
             if tr.perf(aux)<MEJORperf
                 MEJORperf=tr.perf(aux);
                 PESOSent=net.IW{1,1};
                 PESOSsal=net.LW{2,1};
                 PESOSfijos=net.b{1,1};
                 PESOSfijos2=net.b{2,1};
```

```
            perf=tr.perf(aux);
        end

    end
```

```
% Guardamos los resultados finales en "REDESobtenidas33"
save REDESobtenidas33 IW LW bias bias2 PERFORMANCE nodos
```

FIN PROGRAMA "CrearRedesD33"

PROGRAMA "VERRED33"

```
% Elegimos el número de la red que queremos ver (están numeradas por
orden de creación)
load datos33
load REDESobtenidas33
NUM=input('Número de la red a crear: ');
net=newff(ab,s,nodos);
net.IW{1,1}=IW(:,:,NUM);
net.LW{2,1}=LW(1,:,NUM);
net.b{1,1}=bias(:,1,NUM);
net.b{2,1}=bias2(1,1,NUM);

% Creamos los datos que habíamos dejado para testear con "creardatos"
y los enfrentamos con los valores teóricos
run creardatos33
scatter(Srec,Sff,2.3); hold on; rectaux=Srec; plot(Srec,rectaux);
performance=PERFORMANCE(1,NUM)
```

Subrutina "creardatos33"

```
% Creamos los datos
SFFn=sim(net,[Ab; Bb]);

% Desnormalizamos los datos creados
SFF=10.^((SFFn-0.6).*10)-1e-6;

% Seleccionamos los datos con los que no ha sido entrenada la red
Sff=SFF(n+1:1:N);

% Creamos un vector con los datos teóricos en el mismo orden que los
datos barajados, para enfrentar cada uno con el que corresponda
Srec=S(Db(n+1:1:N));
```

FIN PROGRAMA "VERRED33"

PROGRAMA "ERROR33" _

```
% Una vez vistos los datos creados por la red enfrentados con los
teóricos, vamos a ver el error medio, la mediana y el error máximo con
este programa
% Creamos el vector Verror donde se almacenan los errores en valor
absoluto dato por dato y el vector Verror2, donde se almacenan los
errores con signo, para enfrentarlos luego con el vector X que va
desde 1 hasta N-n
Verror=zeros(1,N-n);
Verror2=zeros(1,N-n);

run creardatos33
for i=1:N-n
    Verror(i)=abs(Sb(i+n)-Sff(i));
    Verror2(i)=Sb(i+n)-Sff(i);
end
ErrorMedio=mean(Verror)
ErrorMax=max(Verror)

% Creamos el vector X y lo enfrentamos con Verror2
X=1:1:N-n;
scatter(X,Verror2,2.3); hold on; plot(X,0);
```

FIN PROGRAMA "ERROR33" _

APÉNDICE C
PRESUPUESTOS

En el presente anexo se presentan los presupuestos del proyecto y de la propuesta de aplicación de redes neuronales artificiales para la resolución del otro problema que hemos elegido.

En primer lugar mostraremos el presupuesto del proyecto (proyecto 1) teniendo en cuenta las horas reales de trabajo empleadas. Gran parte de las horas las he empleado en aprender a manejar las RNA y a usar estas con MATLAB. Este tiempo por lo tanto, no es tiempo íntegro de trabajo y se tendrá en cuenta reduciendo el coste horario.

Después, mostraremos el presupuesto estimando las horas de trabajo que serían necesarias a día de hoy, sin tener que invertir tiempo en formación.

Por último mostraremos el presupuesto para la climatización de una sala de oficinas.

Presupuesto del proyecto 1:

- Fecha de inicio: 1 de octubre de 2011.
- Fecha de finalización: 31 de mayo de 2012.

Teniendo en cuenta los fines de semana, los días festivos y un periodo vacacional del 23 de diciembre de 2011 hasta el 2 de enero de 2012 (ambos incluidos) tenemos un total de 152 días empleados. Considerando una media de 4 horas diarias empleadas, el total asciende a 608 horas de trabajo.

- Coste horario del personal:

Ingeniero Técnico Industrial Mecánico → 25 €/hora

Descontamos el 60% del coste de personal dado que el ingeniero en cuestión se encuentra en fase de aprendizaje, por lo que el coste horario se reduce a 10 €/hora.

- Coste del hardware empleado:

Ordenador portátil HP

modelo DV3-4130SS i5-460M/4GB/320GB/13.3"/HD5470 → 625€

- Coste del software empleado:

Licencia de MATLAB r2012a → 300 €/mes

Toolbox MATLAB redes neuronales → 50 €/mes

Al presupuesto final añadimos un margen de riesgo del 15%, un beneficio total del 10%, quedando de la siguiente manera:

Descripción	Coste
Coste del personal	6.080 €
Coste del software	2.450 €
Coste del hardware	625 €
Riesgo (15%)	1.373,25 €
Beneficio (10%)	915,5 €
IVA (18%)	2.059,88 €
TOTAL	**13.091,65 €**

Presupuesto del proyecto 2:

- Fecha de inicio: 1 de junio de 2012.
- Fecha de finalización: 31 de agosto de 2012.

Teniendo en cuenta los fines de semana, los días festivos y un periodo vacacional del 1 de agosto al 15 de este mes (ambos incluidos) tenemos un total de 43 días empleados. Considerando una jornada de trabajo de 8 horas al día, tenemos un total de 344 horas empleadas.

- Coste horario del personal:

Ingeniero Técnico Industrial Mecánico → 25 €/hora

- Coste del hardware empleado:

Ordenador portátil HP

modelo DV3-4130SS i5-460M/4GB/320GB/13.3"/HD5470 → 625€

- Coste del software empleado:

Licencia de MATLAB r2012a → 300 €/mes

Toolbox MATLAB redes neuronales → 50 €/mes

Al presupuesto final añadimos un margen de riesgo del 15%, un beneficio total del 10%, quedando de la siguiente manera:

Descripción	Coste
Coste del personal	8.600 €
Coste del software	1.050 €
Coste del hardware	625 €
Riesgo (15%)	1.541,25 €
Beneficio (10%)	1.027,5 €
IVA (18%)	2.311,88 €
TOTAL	**14.693,25 €**

Presupuesto del proyecto de climatización de una sala de oficinas:

- Fecha de inicio: 1 de junio de 2012.
- Fecha de finalización: 28 de septiembre de 2012.

Teniendo en cuenta los fines de semana, los días festivos y un periodo vacacional del 1 de agosto al 15 de este mes (ambos incluidos) tenemos un total de 63 días empleados. Considerando una jornada de trabajo de 8 horas al día, tenemos un total de 504 horas empleadas.

- Coste horario del personal:

Ingeniero Técnico Industrial Mecánico → 25 €/hora

Montador de equipos de climatización →15 €/hora

- Coste del hardware empleado:

Ordenador portátil HP

modelo DV3-4130SS i5-460M/4GB/320GB/13.3"/HD5470 → 625€

Equipo de control:

Ordenador → 350€

Bacteria de refuerzo → 200€

Tarjeta GSM → 20€

- Instalación:

Bomba de frío-calor de 60.000 frigorías → 7.000€

3 accionadores ranuras de techo → 1.500€

5 sensores de temperatura → 300€

Material de obra → 700€

- Coste del software empleado:

Licencia de MATLAB r2012a → 300 €/mes

Toolbox MATLAB redes neuronales → 50 €/mes

Al presupuesto final añadimos un margen de riesgo del 15%, un beneficio total del 15%, quedando de la siguiente manera:

Descripción	Coste
Coste del personal	20.160 €
Coste del software	1.400 €
Coste del hardware	1.195 €
Coste de la instalación	9.500 €
Riesgo (15%)	4.838,25 €
Beneficio (15%)	4.838,25 €
IVA (18%)	7.547,67 €
TOTAL	**49.479,17 €**

A este presupuesto, habría que añadirle un coste anual por mantenimiento de la instalación (donde se incluye el proceso de optimización del sistema de RNA), que se estima en 150 € (IVA incluido).

REFERENCIAS BIBLIOGRÁFICAS

- Kevin L. Priddy and Paul E. Keller, “Artificial Neural Networks, An Introduction”.

- V. Tribaldos, “Phys. Plasmas 8, 1229 (2001)".

- W. I. van Rij and S. P. Hirshman, “Phys. Fluids B 1, 563 (1989)"

- Xabier Basogain Olabe, “Redes neuronales artificiales y sus aplicaciones”.

- Marcos Gestal Pose, “Introducción a las redes de neuronas artificiales”.

- http://www.mathworks.es/

- http://www-fusion.ciemat.es

- http://www.astro.puc.cl

- http://www.cienciapopular.com